Auton Agent Multi-Agent Syst (2010) 20:308–341
DOI 10.1007/s10458-009-9086-9

Coordination by design and the price of autonomy

Adriaan ter Mors · Chetan Yadati · Cees Witteveen · Yingqian Zhang

Published online: 8 April 2009

Abstract We consider a multi-agent planning problem as a set of activities that has to be planned by several autonomous agents. In general, due to the possible dependencies between the agents' activities or interactions during execution of those activities, allowing agents to plan individually may lead to a very inefficient or even infeasible solution to the multi-agent planning problem. This is exactly where *plan coordination methods* come into play. In this paper, we aim at the development of *coordination by design* techniques that (i) let each agent construct its plan completely independent of the others while (ii) guaranteeing that the joint combination of their plans always is coordinated. The contribution of this paper is twofold. Firstly, instead of focusing only on the feasibility of the resulting plans, we will investigate the additional *costs* incurred by the coordination by design method, that means, we propose to take into account the *price of autonomy*: the ratio of the costs of a solution obtained by coordinating selfish agents versus the costs of an optimal solution. Secondly, we will point out that in general there exist at least two ways to achieve coordination by design: one called *concurrent decomposition* and the other *sequential decomposition*. We will briefly discuss the applicability of these two methods, and then illustrate them with two specific coordination problems: coordinating tasks and coordinating resource usage. We also investigate some aspects of the price of autonomy of these two coordination methods.

Keywords Multi-agent systems · Coordination · Autonomous planning · Algorithms

A. ter Mors · C. Yadati · C. Witteveen · Y. Zhang (✉)
Faculty of Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands
e-mail: Yingqian.Zhang@tudelft.nl

C. Yadati
e-mail: C.Yadati@tudelft.nl

C. Witteveen
e-mail: C.Witteveen@tudelft.nl

A. ter Mors
Almende, Westerstraat 50, 3016 DJ Rotterdam, The Netherlands
e-mail: adriaan@almende.org; A.W.Termors@tudelft.nl

1 Introduction

Multi-agent planning has received a great amount of attention in AI and the agent community. Intuitively, a multi-agent planning problem refers to making a plan or schedule (or a set of plans/schedules) for a set of activities for and by several agents. Usually, due to constraint specifications for these activities, it is not possible for these agents to make these plans completely independently from the other agents.

A well-known everyday example is *trip planning* (especially in the holiday season): suppose several agents want to drive from their current source to their chosen destinations, using some common traffic infrastructure. This requires planning suitable routes. If they make their plans individually, these agents cannot always be guaranteed that their individual plans are jointly conflict-free: often, the finite capacity of the road infrastructure will prevent them from using the same route segment at the same time and we need traffic management systems to remove those conflicts.

A similar problem occurs in *patient scheduling* in hospitals. Here, the agents might be doctors that suggest specific sequences of treatments for their patients (using the CT scan, taking blood samples, examination by colleagues, etc.). If, however, a patient is treated by two or more doctors, this might easily result in a conflict if one doctor suggests to do X immediately before Y while another suggests to do first Y, then Z, and finally X. Here, the autonomous and independent planning activities of the agents could easily result in an infeasible treatment plan.

In general, due to the possible dependencies between agents' tasks or interactions during execution of those tasks, allowing agents to plan individually may lead to a very inefficient or even infeasible solution of such multi-agent planning problems. To handle these dependencies two different, but related, approaches can be distinguished: *plan coordination* methods and *plan decomposition* methods.

Plan coordination. In general, a plan coordination method should ensure that individually proposed plans always result in a globally feasible plan. Plan coordination methods have been studied quite extensively in the multi-agent community [18]. One main approach is the *plan merging* or *plan fusion* (cf. [14,21,57,54]) approach, where coordination is applied after plans have been developed. Here, it is assumed that agents independently work on their own part of the planning problem and achieve a solution for it. Then, in an after-planning coordination phase, possible conflicts between these independently generated individual plans are resolved and positive interactions between them are exploited by exchanging and revising parts of the individual plans. Note that such a plan merging process requires either a centralized processing of distributed plans or communication between agents and information-sharing of some parts of the developed plans. Moreover, the planning agents themselves should be willing to revise their initial plans after conflicts have been detected.

Another main approach is the *coordination during planning* approach (cf. [15,19,20,34,45]). Here, coordination and planning are treated as *intertwined* processes where the agents continuously exchange planning information to arrive at a joint solution. From a coordination perspective, the main difference with the first plan merging approach is that positive (negative) interactions between *partial* individual plans are exploited (resolved) *before* an agent comes up with a completely developed plan. Viewing the plan merging process as a kind of (plan) filtering process, this approach can be seen as an application of the well-known *filter promotion*[1] technique in programming [5]. Note that also in this coordination approach

[1] Filter promotion refers to a programming technique to turn a generate-and-test mechanism into an efficient algorithm by pushing (promoting) the tester into the generator.

agents have to be cooperative in the sense that they should be willing to exchange planning information with other agents and change their current plans, if necessary. On an implementation level, several methods for realizing such concurrent coordination processes with communication exist. For instance, Marecki [45] in his DEFACTO system uses *proxies* [44] where coordination is brought about using a system of tokens.

Whereas the first two coordination approaches use coordination as a filter after (partial) plans have been generated, in a third approach the filter has been placed *before* the plan generation process: in this *coordination by design* approach we aim at the development of coordination techniques that (i) let each agent construct its plan completely independently from the others, thereby (ii) guaranteeing that the joint combination of their plans always is coordinated. In this coordination process some or all of the dependencies between the agents are resolved before any planning takes place. The most influential of such *pre-planning* coordination approaches in the literature are *social laws* and *cooperation protocols*. Social laws (cf. [37,48]) are general rules that govern the agent's behavior; if a collection of agents abide by these rules, then their behavior will be coordinated without the need for any problem-specific information exchange between the agents. In many situations, however, coordination cannot be achieved (or not efficiently) through general, problem-independent rules alone. In such cases, *cooperation protocols* [29] can be applied. Such protocols require simple forms of problem-specific information exchange before the agents can start planning, and they guarantee that if the agents adhere to the protocol, then the individual plans can easily be assembled into a joint plan for the overall task. Examples of such coordination by design strategies are the Temporal Decoupling Method by Hunsberger [27,28] and the pre-planning coordination method discussed in [10]. In these methods, additional constraints are imposed on a set of tasks given to a collection of agents to ensure that they can plan autonomously, while still ensuring that the plan of every agent will satisfy the original set of constraints. Especially in [10], the main focus is on the complexity issues associated with finding a *minimal* set of additional constraints to implement such a coordination by design approach. Finally, viewing coordination by design from a broader perspective, two other relevant lines of research on coordination by design should be mentioned. The first is in *organizational design* where constraints are imposed on agents in order to make local decisions that fit together [50]. The second is in Robocup [52], where constraints are imposed in the form of role specifications to ensure that agents make local decisions that do not conflict.

Plan decomposition. Instead of viewing coordination as a process induced by the way the agents are allocated to subproblems, plan decomposition tries to come up with a decomposition of the problem into subproblems. On the basis of a decomposition an allocation of subproblems to agents can be suggested and a way to coordinate the solutions (plans) of these subproblems. Decomposition is a well-known approach in mathematics and computer science to solve a problem into smaller pieces (sub-problems). It solves these smaller parts, and then obtains the solution to the original problem by combining the solutions to the parts. In principle, such a decomposition offers the possibility to solve the subproblems by several agents concurrently. In AI-planning, however, examples like [31] where plan decomposition results in completely independent subproblems are rather rare. In general, plan decomposition methods [46] do not necessarily aim at decomposing the planning problem into *independent* subproblems: interactions between the different subplans developed are allowed.

For example, in *localized planning* [32,33], the problem is decomposed into so-called regions (subproblems) requiring localized interaction during planning. Here, each region constitutes a subproblem that is solved by an individual agent. The outcome of the decomposition can be used to specify exactly where subproblems will interact (the local interaction regions) and the agent has to communicate to other agents to avoid (negative) interactions

with other planning agents. Therefore, localized planning could also be viewed as a plan coordination during planning method.

While localized planning can be seen as a horizontal decomposition, other approaches can be better viewed as *vertical* decompositions of the problem. In these latter approaches one decomposes a planning problem into multiple levels of abstraction [11,12,14] in order to identify interactions between actions. In this way conflicts might be resolved at higher abstraction levels, ignoring details that only appear at lower levels. Except for single agent applications, such a decomposition can also be used in a multi-agent planning context to resolve possible interactions between agents at several levels. This approach can also be viewed as a coordination during planning approach.

Summarizing, we observe that except for the coordination before planning and complete plan decomposition methods, in order to find a solution to the complete planning problem by applying plan coordination or plan decomposition, one needs to assume that the agents are willing or able to communicate in order to achieve a coordinated result. In this paper we do not want to assume that the agents are cooperative in the sense that they are willing to communicate parts of their plans or revise them if such is essential. Therefore, we will concentrate on a coordination by design approach, where the goal is to design a set of (coordination) rules such that the planning agents (players) will achieve a joint goal no matter which individual plan they choose.

Remark 1 Although such a plan coordination by design could also be seen as a form of *mechanism design* [39], in designing the plan coordination mechanism one does not need to take into account the effect particular combinations of strategies (plans) have on the quality of the resulting joint plan. It is because in general, utilities of plans and plan steps will be unknown to the (other) agents.

In the remainder of this paper, the focus will be on a further analysis of the coordination by design approach. However, instead of merely concentrating on coordination as a method to guarantee the joint *feasibility* of the resulting plans, first of all we will investigate the additional *costs* incurred by the coordination by design method. That is, we propose and illustrate the use of a cost measure (the *price of autonomy*) that is used to measure the overhead of using a coordination method to ensure conflict-free planning or scheduling by selfish agents. This cost measure will be defined as the ratio of the worst case costs of a plan achieved by a coordination mechanism for selfish agents versus the costs of an optimal plan.

Secondly, we will point out that, in general, there exist at least two ways to achieve coordination by design: one called *concurrent decomposition* and the other *sequential decomposition*. In brief, the first technique achieves coordination by adding a set of additional constraints for each agent simultaneously before they start to plan. However, in many cases it is very difficult or even impossible to apply this concurrent decomposition strategy. For example, if the plan tasks given to the agents require access to a common set of scarce resources, mutual exclusion requirements might incur conflicts in the execution of a given set of independently constructed plans. Or, consider the case where the tasks themselves are shared between the agents. Also in this case, concurrent decomposition would be hard to achieve unless the planning freedom of the agents is severely restricted. Hence, instead of the concurrent decomposition, in such cases we might apply an alternative so-called *sequential decomposition* approach. Here, the coordination mechanism allows the agents to plan one by one, ensuring that the plans of all predecessors of a planning agent A_i have been translated into constraints for agent A_i, ensuring that those plans cannot be invalidated by any plan that A_i is able to come up with. As a result, like in the concurrent decomposition approach, the total set of plans developed by the agents is guaranteed to be conflict-free.

Table 1 Transportation orders and tasks

Item	Locations	Tasks
i_1	A, C, B	$t_1 = (\text{A, C}) \prec t_2 = (\text{C, B})$
i_2	C, B, A	$t_3 = (\text{C, B}) \prec t_4 = (\text{B, A})$
i_3	B, A, C	$t_5 = (\text{B, A}) \prec t_6 = (\text{A, C})$

To provide the reader with some intuition about both these plan coordination problems that we will discuss in the subsequent sections and the concurrent and sequential decomposition approaches to solve them, first we illustrate our ideas with a simple transportation example.

1.1 A transportation example

Consider a transportation problem where *items* (packages, goods, etc.) have to be delivered to a sequence of locations by transportation agents. None of the agents can reach all the locations in the infrastructure, so the transportation of some packages may involve transferring a package from one agent to another. Some of the infrastructure resource can have a limited capacity in terms of the number of agents that can simultaneously make use of the resource.

Table 1 defines a transportation planning problem, where three items are required to be delivered: item i_1 needs to be delivered from location A to location C (defined as task t_1) and then from C to B (task t_2); item i_2 from location C to location B (task t_3) and then from B to A (task t_4); item i_3 from B to A (task t_5) and then from A to C (task t_6). The infrastructure is shown in Fig. 1a, all the roads (edges) r_i ($i = 1, \ldots, 6$) and location D have capacity $c()$ equal to 1, whereas locations A, B, and C have unbounded capacity. There are three transportation agents: A_1, starting at location A and operating on the left side of the infrastructure (that is, in Fig. 1a, A_1 can reach locations A, C, and D, and roads r_1, r_4, and r_5); A_2 starting in location C and operating in the right side of the infrastructure (it can reach locations C, B, and D, and roads r_2, r_6, and r_5); and A_3 starting in B and operating in the top part of the infrastructure (it can reach locations B, A, and D, and roads r_3, r_4, and r_6). The travel durations $d()$ are 1 for locations A, B, C, and D; 2 for roads r_4, r_5, and r_6; 6 for road r_1; 7 for road r_2; and 3 for road r_3.

Given that the agents can only access a part of the infrastructure, only the following task allocation is a valid one: agent A_1 gets t_1 and t_6, agent A_2 gets t_2 and t_3, and agent A_3 gets t_4 and t_5. Figure 1b represents the task structure and the allocation to the agents. The system's overall performance is measured by the makespan of completing the global plan. To show how to apply the concurrent and sequential decomposition coordination methods in this transportation instance, we will consider the following two cases: (1) to ignore resource constraints, and (2) to ignore task constraints.

Case 1 If we ignore resource constraints but take into account the precedence constraints between tasks, then the heart of the problem is to coordinate the task planning of the agents. Figure 1b and Table 1 represent this task coordination problem. Each agent has two tasks, one corresponding to a *first-stage* delivery of a package to its intermediate location, the other corresponding to a *second-stage* delivery. Note that both tasks have the same trajectory. Since we ignore resource constraints in this case, all roads and locations can be used by more than one agent simultaneously. However, we assume that every agent can only deliver one item from one location to the other location at each time.

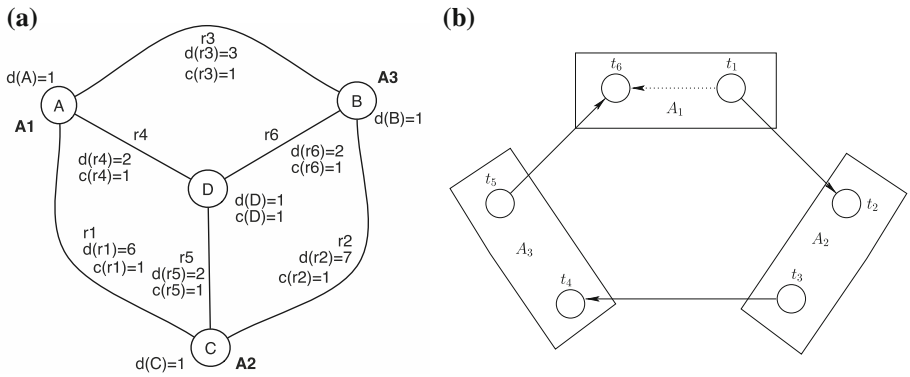

Fig. 1 Transportation problem where agents have to deliver packages to a sequence of locations. **a** Infrastructure of roads (*edges*) and locations (*nodes*), where $d(\cdot)$ denotes duration, i.e., travel time, and $c(\cdot)$ defines capacity. **b** Task graph: *circles* represent tasks, *arrows* are precedence constraints, and *rectangles* specify the allocation of tasks to agents; coordination can be achieved if, for example, agent A_1 adds local constraint $t_1 \prec t_6$

The concurrent coordination method works as follows: First, it decomposes the problem into three sub-problems where each agent needs to plan its own tasks (i.e., which item to deliver first. See Fig. 1b). Then the coordination method should specify some additional constraints on some agent in order to ensure that a feasible solution is *always* guaranteed when three independently solved plans by three agents are combined. From Fig. 1b, observe that if any agent would perform its first-stage task before its second-stage, then no deadlocks can occur, regardless of the order in which the other agents perform their tasks. Thus, a coordination method can add *any* of the following constraints to ensure the feasibility of the global solution: (1) $t_5 \prec t_4$ on agent A_3; or (2) $t_1 \prec t_6$ on agent A_1, or (3) $t_3 \prec t_2$ on agent A_2. If the coordination method enforces all these three constraints on three agents, an optimal solution can be achieved. However, as we have stated earlier, we would like to keep such restrictions on agents as minimal as possible. Hence, the question here is, when the coordination method chooses a *specific* agent to impose the additional constraint, what is the impact on the system's performance?

(i) Let a coordination method M_1 add the constraint $t_5 \prec t_4$ on agent A_3. Note that agents A_1 and A_2 are free to plan their own tasks as long as their plans satisfy the original task constraints and the additional constraint imposed by M_1. When they both decide to deliver their first-stage tasks (i.e., t_1 and t_3) first, then to go back and perform the second-stage tasks (t_6 and t_2), then these agents can execute their tasks simultaneously. Notice that the shortest routes for A_1 and A_2 are those via location D. When all three agents plan their own routes optimally, this results in a merged plan that is *optimal* with minimal makespan of $7 \times 3 = 21$ (i.e., the task completion time of A_1 or A_2). However, the worst case on makespan appears if A_1 and A_2 both plan to do their second-stage tasks, and then the first-stage tasks. Every agent then needs to wait until the other agent finishes its first-stage task. This leads to a sequential task execution: $t_5 \rightarrow t_6 \rightarrow t_1 \rightarrow t_2 \rightarrow t_3 \rightarrow t_4$. Hence the combined plan has a makespan of $5 + 7 \times 3 + 7 \times 3 + 5 = 52$.

(ii) Now consider another coordination method M_2 which adds the constraint $t_1 \prec t_6$ on agent A_1. Same to the situation (i), it will also result in the optimal plan with makespan of 21 when A_2 and A_3 deliver their first-stage tasks first, i.e., $t_3 \prec t_2$ and $t_5 \prec t_4$. The worst case, due to the autonomy of agents A_2 and A_3, is when A_2 and A_3 plan their second-

stage tasks first, i.e., $t_2 \prec t_3$ and $t_4 \prec t_5$. If this happens, the makespan of the resulting global plan is 50, with a task execution sequence $t_1 \rightarrow t_2 \rightarrow t_3 \rightarrow t_4 \rightarrow t_5 \rightarrow t_6$.

Notice that if we impose the constraint $t_3 \prec t_2$ on agent A_2, the resulting worst-case makespan is the same to that of the situation (ii) by M_2. Both of the coordination methods M_1 and M_2 guarantee the feasibility of the global solution. However, in this example, it is not difficult to see that M_2 is preferable to M_1, since M_2 provides a better *worst case makespan* than M_1.

Case 2 If we ignore task constraints but not resource constraints, we are effectively assuming that the second-stage packages are available for the agents from the beginning. For simplicity, we consider that the task of each agent is to deliver one item from one location to the other, i.e., agent A_1 delivering from A to C; agent A_2 from C to B; and agent A_3 from B to A. In this case, agents only need to drive once from their pickup location to their delivery location. However, we do need to ensure that *only one* agent may make use of the capacitated resources r_4, r_5, r_6, and D at the same time.

In this resource usage coordination case, the concurrent decomposition approach is inapplicable, as all agents may (individually) plan to use the location D during same period, which incurs the conflicts on the resource usage. Therefore, we apply the sequential decomposition method here. The different sequential coordination methods may ask the agents to plan sequentially in different orders, i.e., one allows A_1 to plan first, then A_2 after it receives the updated plan from A_1, and finally A_3 after A_2 communicates its plan; or another asks A_2 to plan first, followed by A_1, and then A_3. We show the influence of the agent planning order as follows.

(i) Assuming agent A_1 is chosen by a sequential coordination method M_1' to be first to plan its use of resource. It comes up a plan which takes the shortest route to C (i.e., along r_4, D, and r_5) as: $\{(A, [0, 1]), (r_4, [1, 3]), (D, [3, 4]), (r_5, [4, 6]), (C, [6, 7])\}$, where $(A, [0, 1])$ denotes that A_1 traverses at location A from time 0 to time 1; $(r_4, [1, 3])$ specifies it travels along the road r_4 during time 1 and 3, etc. This plan induces an additional constraints on the resource use to agent A_2, because if A_2 takes the shortest route to B, then it will have conflict with A_1 at location D during time step $[3, 4]$. Thus, A_2 takes the "detour" along r_2 which takes travel time of 7, and completes its delivery at time step 9. Finally, agent A_3 comes up a plan to take its shortest route r_3 to A, respecting the constraints introduced by agents A_1 and A_2. Note the combined plan of three agents results in a makespan of 9.

(ii) The makespan of the global plan can be improved by coordination method M_2', which lets agent A_2 plan first. A_2 takes the shortest route to its destination B along r_5, D, and r_6 and completes its task at time 7. Due to the conflict of using location D, A_1 has to take detour along r_1 to location A, which requires travel time of 6. A_1 can finish its task by time 8. A_3 travels through its shortest route r_3 to A with total time 5. As a result, the combined plan leads to a makespan of 8, which is optimal.

In this example, although both M_1' and M_2' guarantee feasible global plans, we favor M_2' over M_1' because M_2' results in a better makespan (i.e., 8) than M_1' (i.e., 9).

The above transportation example shows how the concurrent and sequential coordination methods can be used in a transportation domain. The example also made clear that feasible coordination methods might differ in efficiency and hence they could have a different price of autonomy. We will study the price of autonomy in more detail on two different approaches to coordination by design: a task coordination framework (using concurrent decomposition

method) in Sect. 3, and a resource coordination framework (using sequential decomposition method) in Sect. 4. Before introducing these two specific frameworks, in the next section we present a general coordination by design framework.

2 Coordination by design: a general framework

As we argued in the introduction, in a (task-based) multi-agent planning problem, a set of tasks is given to a set of agents. These agents have to complete such tasks by making a joint plan to complete them. Usually there is also a (common) set of resources needed to complete such tasks. Both tasks and resources are subject to some sets of constraints.

To model such a multi-agent task-based planning problem, we consider a tuple $\Theta = (\mathcal{A}, \mathcal{T}, R, C, \phi, \psi)$, where $\mathcal{A} = \{A_1, \ldots, A_n\}$ denotes a set of agents, $\mathcal{T} = \{t_1, \ldots, t_m\}$ is a set of tasks, $R = \{r_1, \ldots, r_p\}$ is a set of resources and C is a non-empty set of constraints on tasks and resources. Usually, the set of constraints C is partitioned into $C = C_{\mathcal{T}} \cup C_R$ where $C_{\mathcal{T}}$ is the set of constraints on the tasks and C_R is the set of constraints on the resources. The functions $\phi : \mathcal{A} \to 2^{\mathcal{T}}$ and $\psi : \mathcal{T} \to 2^R$ are the *task-assignment* function and the *task-resource* function, respectively. Here, the task-assignment function specifies which agent is assigned to which collection of tasks.[2] The task-resource function specifies which collection of resources is needed to execute which task. In the general framework, we do not make any further assumptions on task and resource constraints, or task-assignment and task-resource functions. Instead, they are dependent on specific applications. The following example shows a detailed specification of a transportation planning problem.

Example 1 Consider the transportation example in Sect. 1.1. Here, there is a set of tasks $\mathcal{T} = \{t_1, t_2, t_3, t_4, t_5, t_6\}$ allocated to the set of agents $\mathcal{A} = \{A_1, A_2, A_3\}$ by the allocation function ϕ specified as $\phi(A_1) = \{t_1, t_6\}$, $\phi(A_2) = \{t_2, t_3\}$ and $\phi(A_3) = \{t_4, t_5\}$. The set of task constraints $C_{\mathcal{T}}$ defines a set of precedent relations between tasks: $C_{\mathcal{T}} = \{t_1 \prec t_2, t_3 \prec t_4, t_5 \prec t_6\}$.

The available resources are a set of locations $R = \{A, B, C, D, r_1, r_2, r_3, r_4, r_5, r_6\}$. The resource constraints C_R specify the capacity constraints of each location: $C_R = \{c(r) \leq v | r \in R, v \in \mathbb{N} \cup \{\infty\}\}$, and we have $c(A), c(B), c(C) = \infty$ and $c(r_1), c(r_2), c(r_3), c(r_4), c(r_5), c(r_6), c(D) = 1$.

Finally, the task-resource function ψ defines the resource usage of each task, $\psi(t_1) = \psi(t_6) = \{A, C\}$, $\psi(t_2) = \psi(t_3) = \{C, B\}$, $\psi(t_4) = \psi(t_5) = \{B, A\}$.

Here, the task constraints define a partially ordered precedence relation between tasks. The resources, i.e., the locations, are non-consumable, and mutually exclusive according to the capacity constraints.

A *solution* to a multi-agent planning problem $\Theta = (\mathcal{A}, \mathcal{T}, R, C, \phi, \psi)$ is a specification of a *task-resource plan* (or plan, for short). Such a plan is a tuple $P = (\mathcal{A}, \mathcal{T}, R, C^P, \phi, \psi)$, where C^P is a non-empty set of *plan constraints* refining C, that is, if all constraints in C^P are satisfied, then C is satisfied too. To preserve the generality of the task planning framework, we will not provide an exact specification of C^P. We only have to require that C^P is consistency preserving, i.e., if C can be satisfied then C^P can, too.

Example 2 Consider Case 1 (ignoring the resource constraints) in the transportation example (Fig. 1a, b). The original constraints are a set of task constraints $C = C_{\mathcal{T}} = \{t_1 \prec t_2, t_3 \prec$

[2] We do not consider in this paper the problem of task assignment or task allocation, which has been studied extensively (cf. [47]).

$t_4, t_5 \prec t_6\}$. The plan P shown in the example of Case 1, using coordination method M_2, specifies the execution order of the tasks of all the agents, which can be represented by the following plan constraints: $C^P = \{t_1 \prec t_6, t_5 \prec t_4, t_3 \prec t_2, t_1 \prec t_2, t_3 \prec t_4, t_5 \prec t_6\}$. Here, C^P induces exactly one plan of executing the tasks, and it is a refinement of the original constraints C (i.e., $C^P \supseteq C$).

Coordination by design requires that from the specification of a multi-agent planning problem Θ, we are able to derive a set $\{\Theta_i\}_{i=1}^n$ of single agent planning problems that can be solved independently. Using the framework, we say that a multi-agent problem $\Theta = (\mathcal{A}, \mathcal{T}, R, C, \phi, \psi)$ *induces* a single agent planning problem $\Theta_i = (T_i, R, C_i, \psi)$ for agent A_i, where $T_i = \phi(A_i)$ is the set of tasks assigned to A_i, and $C_i \subseteq C$ is the set of constraints *restricted* to tasks occurring only in T_i. That is, in a single agent plan, only the constraints exclusively to agent A_i are specified, and any shared constraints between A_i and other agents are discarded. Like the solution for the overall planning problem, the solution of the single agent planning problem Θ_i is a single agent plan $P_i = (T_i, R, C_i^P, \psi)$ where C_i^P is a set of plan constraints that refine C_i.

We say that the problem Θ is *decomposable* if the set of local plans (individual solutions) $\{P_1, \ldots, P_n\}$, can be combined into a global plan $P = \sqcup P_i = (\mathcal{T}, R, \bigcup_{i=1}^n C_i^P, \phi, \psi)$, and P is a solution to the original problem Θ.

In general, however, a multi-agent planning problem Θ will not be decomposable and we will need a *coordination mechanism* to ensure that the local plans can be combined into a feasible global plan.

Such a coordination (by design) mechanism M is an algorithm that, given as input a multi-agent planning problem $\Theta = (\mathcal{A}, \mathcal{T}, R, C, \phi, \psi)$, an agent A_i, and a set $\mathcal{P}_{-i}$ of plans of other agents, not including the plan of agent A_i, returns a single agent planning problem Θ_i^M such that:

1. $\Theta_i^M = (T_i, R, C_i^M, \psi)$ is an individual planning problem for agent A_i, where $T_i = \phi(A_i)$;
2. A_i is allowed to come up with whatever solution (plan) P_i^M for Θ_i^M;
3. the combination $P = P_i^M \cup \mathcal{P}_{-i}$ is always a solution to the original problem Θ.

In order to avoid circular dependencies, where a coordination mechanism would need the specification of the plan of agent A_i in order to determine Θ_j^M, but also needs the plan of agent A_j in order to determine Θ_i^M, we call a mechanism M *feasible* if such circular dependencies do not occur. That is, M is feasible for Θ if there exists an enumeration $\langle A_1, A_2, \ldots, A_n\rangle$ of $\mathcal{A}$ such that for every $i = 1, \ldots, n$, it holds that $M(A_i, \Theta, \mathcal{P}_{-i}) = \Theta_i^M$ and $\mathcal{P}_{-i}$ does not include any plan for an agent A_k for any $k \geq i$.

Clearly, if a mechanism M is feasible, then there exists at least one ordering of agents that allow them to plan independently from the others.

Example 3 Consider the coordination method M_2 proposed for the Case 1 in the transportation example. M_2 introduces only one local constraint to agent A_1: $C_1^{M_2} = \{t_1 \prec t_6\}$, while $C_2^{M_2} = C_3^{M_2} = \varnothing$. Note that here the mechanism M_2 is able to produce a local planning problem $\Theta_i^{M_2}$ without the need to know any of the plans specified by the other agents. As a result of $C_1^{M_2}$, the plan of A_1 (i.e., P_1) executes t_1 before t_6. One can verify that no matter what local plans P_2 and P_3 agents A_2 and A_3 come up with, the combined plan constraint $P_1 \cup P_2 \cup P_3$ always implies a feasible solution (a deadlock free joint plan), to the example in Fig. 1b.

2.1 Concurrent and sequential decomposition

As we discussed in the introduction, coordination by design is a plan coordination method that allows agents to plan independently from the other agents. It achieves so by decomposing the original multi-agent planning problem into individual planning problems that can be solved independently.

We now distinguish two main methods to obtain such a decomposition: *concurrent decomposition* and *sequential decomposition*.

Concurrent decomposition. In concurrent decomposition the coordination mechanism M is able to specify—given the original problem Θ and an agent A_i—the set of local problems Θ_i directly without knowing the plans of the other agents. That is, for every agent A_i, the set of plans of other agents that M needs in order to determine the local planning problem for agent A_i is empty and we have $M(A_i, \Theta, \mathcal{P}_{-i}) = M(A_i, \Theta, \varnothing) = \Theta_i^M$. It is immediate that this mechanism is feasible for every enumeration of agents. Hence, each of the agents is able to make its plan independently from and concurrently with the others.

Concurrent decomposition can be applied if (i) the task allocation ϕ induces a partitioning of the tasks, i.e., no task is assigned to more than one agent, and (ii) there is no resource dependency between tasks assigned to different agents, i.e., if $t \in T_i$ and $t' \in T_j$ then $\phi(t) \cap \phi(t') = \varnothing$. In such multi-agent planning frameworks, the set of resources does not play a role in the decomposition process and we obtain a simple plan coordination framework $(\mathcal{A}, \{T_i\}_{i=1}^n, C_{\mathcal{T}})$ where $\{T_i\}_{i=1}^n$ is the partitioning of the set of tasks induced by ϕ and $C_{\mathcal{T}}$ is the set of constraints for the tasks. This framework has been studied in [10,51].

Example 4 Consider again the transportation example. When the resource constraints are disregarded, we end up with a simple plan coordination framework $(\mathcal{A}, \{T_i\}_{i=1}^3, C_{\mathcal{T}})$, where $C_{\mathcal{T}} = \{t_1 \prec t_2, t_3 \prec t_4, t_5 \prec t_6\}$, and the task partitioning is shown in Fig. 1b. The concurrent decomposition coordination method M_2 specifies a local planning problem for agent A_1, that is, $M_2(A_1, \Theta, \varnothing) = \Theta_1^{M_2} = (T_1, C_1^{M_2}, \psi)$, where $T_1 = \{t_1, t_6\}$, $C_1^{M_2} = \{t_1 \prec t_6\}$. It is easy to see that the only plan that A_1 can generate is to execute t_1 before t_6.

Sequential decomposition. On the other hand, in sequential decomposition, there is an enumeration $\langle A_1, A_2, \ldots, A_n \rangle$ of the agents such that for all i, $M(A_i, \Theta, \mathcal{P}_{-i}) = M(A_i, \Theta, \{A_1, \ldots, A_{i-1}\}) = \Theta_i^M$. That is, all plans of the agents A_j preceding A_i in the enumeration are needed to guarantee that agent A_i can plan independently. The idea is that for every i, the results of all the plans P_j of the agent A_j $(j < i)$ are translated, by the coordination mechanism M, into a suitable set of constraints, in such a way that plans of agents $A_i, A_{i+1}, A_{i+2}, \ldots, A_n$ never can invalidate the plans P_j of agent A_j $(j < i)$.

Sequential decomposition can be applied if there is an overlap in the tasks assigned to different agents or there are some dependencies on the resources that have to be used by two agents in the system. If we focus on the resource constraints but neglect the task constraints in the framework, we achieve a *resource coordination* framework $(\mathcal{A}, \mathcal{T}, R, C_R)$. Here, the planning of the tasks of the agents is trivial, because there are no restrictions among them. On the other hand, coordination is needed because of resource dependencies.

Example 5 Consider Case 2 in the transportation example of Sect. 1.1, where if we ignore the task constraints in the example, there are no precedence relations between any tasks assigned to different agents. Hence, we end up with a resource coordination framework $(\mathcal{A}, \mathcal{T}, R, C_R)$, where $\mathcal{A}$, $\mathcal{T}$, R, and C_R are explained in the earlier example (see Example 1).

The coordination method M_1' described in the transportation example demonstrates how the sequential decomposition works. M_1' first defines the local planning problem $\Theta_1^{M_1'}$ for

agent A_1, i.e., $\Theta_1^{M_1'} = M_1'(A_1, \Theta, \mathcal{P}_{-1}) = M_1'(A_1, \Theta, \varnothing)$. Thus, agent A_1 freely makes an optimal local plan P_1 for itself that takes the shortest route along with r_4, D and r_5. P_1 can be represented by the plan constraints: $C_1^P = \{(A, [0, 1]), (r_4, [1, 3]), (D, [3, 4]), (r_5, [4, 6]), (C, [6, 7])\}$, where $(A, [0, 1])$ denotes that A_1 traverses at location A from time 0 to time 1, etc.

Next, M_1' specifies the local planning problem $\Theta_2^{M_1'}$ for agent A_2: $\Theta_2^{M_1'} = M_1'(A_2, \Theta, \{A_1\})$. That is, when A_2 makes its plan, it must respect the plan constraints generated by agent A_1. Due to the plan constraint $(D, [3, 4])$ of A_1, A_2 has to take a longer route and comes up with a plan P_2 which is represented by the following plan constraints: $C_2^P = \{(C, [0, 1]), (r_2, [1, 8]), (B, [8, 9])\}$.

Finally, the local planning problem for agent A_3 is defined as: $\Theta_3^{M_1'} = M_1'(A_3, \Theta, \{A_1, A_2\})$. Agent A_3 makes the following local plan, respecting the plans generated by the preceding agents A_1 and A_2: $C_3^P = \{(B, [0, 1]), (r_3, [1, 4]), (A, [4, 5])\}$.

Notice here, the ordering of the agents of making plans is: $\langle A_1, A_2, A_3 \rangle$.

2.2 Efficiency issues and the price of autonomy

In both sequential and concurrent decomposition, additional constraints are imposed upon the agents to ensure *feasibility* of the resulting plans. Besides feasibility, however, coordination methods should also be evaluated based on the *efficiency* of the resulting joint plan produced by the agents. That is, we should be able to indicate how much efficiency we lose, due to the fact that in our problem we allow selfish agents to come up with their own plans instead of forcing them to use an optimal plan that could require quite a lot of cooperation between the agents. To this end, we introduce the notion of the *price of autonomy* measuring the costs over the performance loss due to the independent planning of the agents. This price of autonomy ρ_M of using coordination method M w.r.t. efficiency is the worst-case ratio of the efficiency of the joint plan obtained by letting selfish agents make their own plan as efficient as possible, versus the efficiency of the most efficient plan that could be obtained as the solution of the multi-agent problem.

Remark 2 The price of autonomy is closely related to the well-known game-theoretical notion of the *price of anarchy* used to measure the worst-case loss arising from insufficient ability or willingness to coordinate the behaviours of selfish agents [40,42,43]. Technically, the price of anarchy is the ratio of the cost of the worst-case Nash equilibrium in a game versus the cost of an optimal solution to the game.

Example 6 Consider the transportation example in Sect. 1.1. For Case 1, the optimal plan, with constraints $\{t_1 \prec t_6, t_3 \prec t_2, t_5 \prec t_4\}$, has the makespan of 21. However, the worst-case makespan is 52 by coordination method M_1, and 50 by coordination method M_2. Thus, the price of autonomy of M_1 is: $\rho_{M_1} = \frac{52}{21} \approx 2.48$, while the price of autonomy of M_2 is slightly smaller: $\rho_{M_2} = \frac{50}{21} \approx 2.38$.

For Case 2, the sequential coordination method M_2' lets the agent A_2 plan first, which leads to an optimal plan with makespan of 8. Hence, the price of autonomy of M_2' in this example is 1, i.e., $\rho_{M_2'} = 1$. However, when A_1 is chosen to be the first to plan (by coordination method M_1'), the resulting makespan is 9, which leads to the price of autonomy of using M_1' of $\rho_{M_1'} = \frac{9}{8} = 1.125$.

2.3 Problems to solve in the task resource framework

In this paper we study two coordination methods for selfish agents: the concurrent and the sequential decomposition method and the price of autonomy ρ_M of these methods using (1) a task coordination framework $(\mathcal{A}, \{T_i\}_{i=1}^n, C_\mathcal{T})$ (Sect. 3), and (2) a resource coordination mechanism $(\mathcal{A}, \mathcal{T}, R, C_R)$ (Sect. 4), respectively.

In the task coordination framework $(\mathcal{A}, \{T_i\}_{i=1}^n, C_\mathcal{T})$, $C_\mathcal{T}$ may induce some constraints between two sets of tasks T_i and T_j. We aim to develop a concurrent decomposition based coordination method (or concurrent coordination) and aim at makespan minimisation as an efficiency criterion. An immediate question then is: Given a coordination instance, *how hard is the problem of designing an optimal makespan efficient concurrent coordination method?* In other words, does there exist an efficient coordination method that computes for each agent a set of additional constraints in polynomial time, such that it minimizes the global make span of the system, i.e., is there a coordination method that realizes $\rho_M = 1$?

Section 3 proposes such a concurrent coordination method M for a specific task coordination problem, where autonomous agents work together for a makespan efficient schedule. We show that when the agents have unbounded capacity to carry out tasks simultaneously, the price of providing autonomy to agents by M is minimised, i.e, M always ensures the optimal global solution.

As we will see, sometimes designing an optimal coordination method turns out to be intractable. In that case, we prefer a polynomial-time feasible coordination method that gives a reasonable bound on the price of autonomy. This is the case in our task coordination problem, when agents are not able to execute an arbitrary number tasks simultaneously. Here, designing an optimal coordination method is NP-hard. As it turns out, however, there is a polynomial time mechanism guaranteeing that the price of autonomy is bounded by 2.

Section 4 studies a resource coordination problem, where we coordinate the usage of resources that the agents need to perform their tasks. The coordination needs to ensure that the usage of a resource is always less than or equal to its capacity. In this case, concurrent decomposition can't work, but sequential coordination is possible. We solve this problem by letting the agents make their (resource) plans in sequence, and by placing *reservations* on resources to reflect their planned usage. The proposed coordination method M can always ensure the feasibility of the merged local plans. The problem of finding an optimal resource plan is intractable. We show that our coordination method M is not a fixed-ratio approximation scheme by giving an example where M leads to the price of autonomy of $\frac{|\mathcal{A}|}{2}$. Furthermore, we address another interesting question raised in sequential coordination: *what is the optimal agent ordering?* More specifically, *whether the ordering of the sequential decomposition has a great impact on the price of autonomy?* We will investigate this experimentally.

We now start with a discussion of the design of a concurrent coordination method for task coordination problem.

3 Concurrent coordination: autonomous task planning

This section considers a plan coordination framework $(\mathcal{A}, \{T_i\}_{i=1}^n, C_\mathcal{T})$, where the usage of resources have been ignored and task constraints exist. We study a coordination by design method for a multi agent distributed scheduling problem, where the coordination method should guarantee high flexibility of the autonomous agents, while the system's optimal performance (minimal makespan) should be preserved as much as possible. In other words, we aim at minimizing the price of autonomy.

Distributed scheduling has been an active area of research in the past decade. Roughly speaking, one can distinguish between approaches that assume that the participating systems, or agents controlling these systems, are cooperative and approaches that assume that the participating agents/systems are non-cooperative. Examples of the former (classical) approaches are DLS [49], HEFT [55], CPOP [55], ILHA [3] and PCT [36]. All these approaches mainly focus on optimizing some performance criteria such as makespan and required communication between processors. Typically, these approaches assume that the participating systems are (i) fully cooperative in (ii) establishing a single globally feasible schedule for the complete set of tasks.

In quite a lot of applications, however, we simply cannot assume that the participating systems are fully cooperative. For example, in grid applications, jobs often have to compete for CPU and network resources, and each agent is mainly interested in maximizing its own throughput instead of maximizing the global throughput. Thus, several researchers have adopted a game-theoretic approach for solving scheduling problems with non-cooperative agents. For example, Walsh et al. [58] use auction mechanisms to arbitrate resource conflicts for scheduling network access to programs for various users on the internet. Another approach to non-cooperative scheduling is based on negotiation. Recently, Li in his Ph.D. thesis developed both static and learning based dynamic negotiation models for grid scheduling [35].

In both approaches to non-cooperative scheduling, however, the effort is directed at developing a *single global* schedule that meets some criterion. Further, the computation of the final schedule is centralised. In situations where agents are exploring unknown or hostile territory, it might be very restrictive or even impossible to enforce rigid schedules on the agents. In other situations where agents participate in more than one such system, they would require a minimum set of constraints on their schedules rather than a single rigid schedule.

Recently, Hunsberger [26] has developed a temporal decoupling method for Simple Temporal Networks (STNs) to decompose an STN into a number of independent sub-STNs, each of which can be scheduled independently from the others. Although the autonomous scheduling method we will apply is related to his decoupling method, we want to design such decoupled subnetworks directly from a given set of simple constraints and compute the minimum makespan from these constraints, whereas Hunsberger starts from a given STN and does not pay attention to efficiency criteria.

We start by describing the coordination framework. The task constraint function $C_{\mathcal{T}}$ specifies, for each task $t_i \in \mathcal{T}$, its duration: $d(t_i) \in Z^+$. Furthermore, $C_{\mathcal{T}}$ also defines a partially ordered precedence relation $\prec$ between tasks, i.e., $t_i \prec t_j$ indicates that task t_i must be completed before task t_j can start. We use the transitive reduction $\ll$ of $\prec$ to indicate the immediate precedence relation between tasks, i.e., $t_i \ll t_j$ iff $t_i \prec t_j$ and there exists no t_k such that $t_i \prec t_k$ and $t_k \prec t_j$. A directed acyclic graph (DAG) $G = (\mathcal{T}, \ll)$ is used to represent the task structure of $\mathcal{T}$.

In addition to the task constraints, we also assume that there is a function $c : \{1, 2, \ldots, n\} \rightarrow Z^+ \cup \{\infty\}$ assigning to each agent $A_i \in \mathcal{A}$ its *concurrency bound* $c(i)$. This concurrency bound is the upper bound on the number of tasks agent A_i is capable of performing simultaneously. We say that $\langle \{T_i\}_{i=1}^n, \prec, c(), d() \rangle$ is a *scheduling instance*.

Given such a scheduling instance, a global schedule for it is a function $\sigma : \mathcal{T} \rightarrow Z^+$ determining the starting time $\sigma(t_i)$ for each task $t_i \in \mathcal{T}$. We define a *feasible* schedule σ as follows.

Definition 1 A feasible schedule is a schedule that satisfies the following constraints:

1. for every pair $t_i, t_j \in \mathcal{T}$, if $t_i \prec t_j$, then $\sigma(t_i) + d(t_i) \leq \sigma(t_j)$;

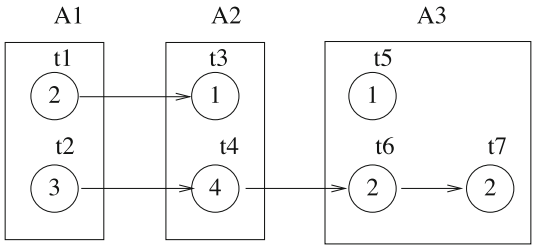

Fig. 2 A set of tasks t_i with their durations $d(t_i)$ to demonstrate the possibility of developing several makespan optimal schedules. Task durations are indicated as numbers within the *circles* representing the tasks

2. for every $i = 1, \ldots, n$ and for every $\tau \in Z^+$, let $s = \{t_j \in T_i \mid \tau \in [\sigma(t_j), \sigma(t_j) + d(t_j)]\}$ we require that $|s| \leq c(i)$, that is the concurrency bounds for every agent A_i should be respected.

An optimal schedule is the one that minimizes *makespan*, i.e., for all feasible schedules σ', the optimal schedule σ satisfies: $\max_{t \in T}\{\sigma(t) + d(t)\} \leq max_{t \in T}\{\sigma'(t) + d(t)\}$.

As for the measure of flexibility of autonomous agents, a coordination method should impose upon each agent a *minimal* set of additional constraints C_i, such that if C_i is specified for the scheduling instance $\langle T_i, \prec_i, d_i(), c(i)\rangle$ of agent A_i, then all *locally feasible* schedules σ_i satisfying their *local* constraints C_i can be merged into a globally feasible schedule σ for the original scheduling instance. The set of constraints C_i that we will add to each local scheduling instance for each agent A_i, is a set of *time intervals* $[lb(t), ub(t)]$ for the tasks in T_i. Each such interval $[lb(t), ub(t)] \in C_i$ with $lb(t), ub(t) \in Z^+$ specifies that any individual schedule σ_i agent A_i might choose has to satisfy the constraint $lb(t) \leq \sigma_i(t) \leq ub(t)$.

Our goal is to design a minimal set of constraints C_i for each agent A_i, such that the merging of individual schedules σ_i that satisfy C_i always is a globally feasible schedule. Moreover, we would also like the merged plan to be makespan efficient. In the next sections, we consider two scenarios: the first, where agents can perform an unlimited number of tasks simultaneously and the second where they can perform only a single task at any given point in time.

Example 7 Consider a simple example shown in Fig. 2, where there are three agents and 7 tasks with precedence constraints. Here, a direct precedence constraint $t \ll t'$ is represented as an arrow from t pointing to t'. The task durations $d(t)$ are indicated within the circles representing the tasks t. Suppose that each agent can perform two tasks simultaneously, that is $c(1) = c(2) = c(3) = 2$. Clearly, the minimal makespan of processing these tasks is 11. To achieve this minimal makespan, the following schedules for the agents are possible: $\sigma_1(t_1) = \sigma_1(t_2) = 0$; $\sigma_2(t_3) = 2$; $\sigma_2(t_4) = 3$ and $\sigma_3(t_5) = 0$; $\sigma_3(t_6) = 7$; $\sigma_3(t_7) = 9$. However, prescribing these schedules is unnecessarily restrictive to the agents. In fact, several other schedules also result in the same global makespan. For example, A_1 could start task t_1 in the interval [0,2] while starting task t_2 in the interval [0, 0]. Agent A_2 can process tasks t_3 in the interval [4, 7] while starting t_4 in [2, 2]. Similarly, agent A_3 can process task t_5 in the interval [8,10] with task t_6 starting in the interval [7, 7] and task t_7 in [9, 9]. Notice here that any schedule produced by the agents such that these intervals are honored will always lead to a global makespan of 11. Thus, by introducing the additional constraints in the form of these intervals, agents have some amount of flexibility on deciding their local schedule without affecting the minimal makespan.

3.1 Coordinating agents with unbounded concurrency

In this section, we show that there exists a surprisingly simple makespan efficient autonomous scheduling algorithm, provided that the agents are capable of processing as many tasks concurrently as required, i.e., they have unbounded concurrent capacity. We first present the proposed coordination algorithm ISA (or, interval-based scheduling algorithm). We then prove that, it provides maximal flexibility to agents and, in addition the solution is optimal. Thus, the price of autonomy is 1.

3.1.1 The ISA *coordination algorithm*

The heart of the ISA algorithm is to specify for every task t, an interval consisting of its earliest possible starting time and its latest possible starting time (without violating $C_{\mathcal{T}}$). Such an interval can be viewed as a constraint on the starting time of a task. Once these constraints are computed, the agents can autonomously create any local schedule provided that it satisfies these constraints. Thus, on the one hand, they are offered the flexibility of creating more than one schedule but on the other they are assured that the global makespan is optimal.

The CPOP [55] algorithm by Topcuouglu et al. uses a similar line of thought. In CPOP, a combined value of the *depth* and the *height* of a task is used to compute the priority for every task. This priority is used later to compute an efficient schedule for the tasks. We build upon this idea and compute intervals instead of priorities, within which each agent is free to schedule its tasks.

To define the interval $[lb(t), ub(t)]$ for the starting time of a task t in the given partial order $\langle T, \prec, d() \rangle$, we first compute, for each task $t \in T$, its $depth(t)$ and its $height(t)$. The depth and the height of a task in a given partial order can be used to determine the earliest and the latest possible time at which a task can be started. To aid in our computation of the depth and height of a task t, we further identify two sets: $pred(t) = \{t' |\ t' \ll t\}$ and $succ(t) = \{t' |\ t \ll t'\}$.

Definition 2 (*Depth of a task t*) $depth(t) = 0$ if $pred(t) = \varnothing$ and $depth(t) = max_{t' \in pred(t)} \{depth(t') + d(t')\}$, otherwise.

Note that the depth of a task t is the maximum duration of any chain of tasks preceding it, hence it directly determines the earliest time task t might start. The depth $depth(T)$ of the set of tasks T is defined as the maximum duration required to complete all tasks taking into account the precedence relation $\prec$: $depth(T) = \max_{t \in T} \{depth(t) + d(t)\}$. So $depth(T)$ defines the minimal makespan of T. The height $height(t)$ of a task t in a partial order $\langle T, \prec, d() \rangle$ defines the time that has to pass before all tasks occurring after t and including t have been completed, thus,

Definition 3 (*Height of a task t*) $height(t) = d(t)$, if $succ(t) = \varnothing$ and $height(t) = max_{t' \in succ(t)} \{height(t') + d(t)\}$, otherwise.

From the specifications of $depth(t)$, $height(t)$ and $depth(T)$ the earliest ($lb(t)$) and latest ($ub(t)$) possible starting times for a task t can be derived as follows: $lb(t) = depth(t)$ and $ub(t) = depth(T) - height(t)$.

These intervals $[lb(t), ub(t)]$, however, cannot be used directly for autonomous scheduling. In general because the length of task chains[3] differ, it can easily happen that the intervals

[3] A task chain is a partial order of tasks, related by the precedence relation $\prec$.

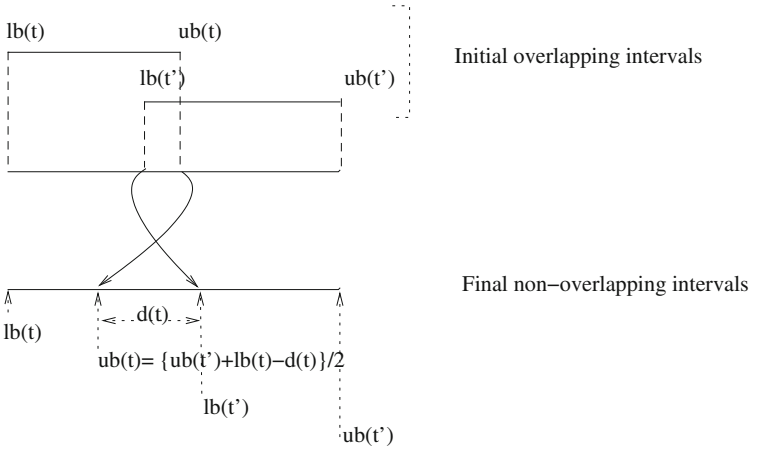

Fig. 3 Overlapping interval splitting procedure in ISA

of some precedence constrained tasks $t \prec t'$ might *overlap*, that is $lb(t') < ub(t) + d(t)$, while $t \prec t'$. Such an overlap might cause a violation of the first constraint on the joint schedule, namely that $t \prec t'$ should imply $\sigma(t) + d(t) < \sigma(t')$.

Example 8 Consider the set of tasks given in Fig. 2. The depth of T is $depth(T) = 11$. Computing the depths and the heights of the tasks t_1 and t_3, we derive the constraints $C(t_1) = [0, 8]$ and $C(t_3) = [2, 10]$. Now, agent A_1 could decide to start t_1 at time $\sigma_1(t_1) = 6$, while A_2 could choose $\sigma_2(t_3) = 5$. However, if these schedules are merged, we violate the precedence constraint between t_1 and t_3 since then $\sigma(t_3) < \sigma(t_1) + 2$.

This implies that, because of overlapping intervals, agents might develop schedules that violate precedence constraints. Therefore, we remove such overlaps as depicted in Fig. 3. In order to satisfy the schedule constraints we should ensure that whenever $t \prec t'$, the difference between $lb(t')$ and $ub(t)$ should be at least $d(t)$.

Note that, in case of an overlap between two tasks $t \prec t'$ belonging to different agents, it is always possible to remove the overlap without creating empty intervals. If $t \prec t'$, we have $lb(t) + d(t) \leq lb(t')$ and $ub(t) + d(t) \leq ub(t')$. The existence of an overlap implies that $lb(t) < lb(t') < ub(t) + d(t)$. Hence, $ub(t') - lb(t) \geq ub(t) + d(t) - (lb(t') - d(t)) > -d(t) + 2d(t) = d(t)$.

Since we require $lb(t') - ub(t) \geq d(t)$, we set $ub(t) = lb(t) + \left\lfloor \frac{ub(t') - lb(t) - d(t)}{2} \right\rfloor$ and thereafter $lb(t') = lb(t) + \left\lfloor \frac{ub(t') - lb(t) - d(t)}{2} \right\rfloor + d(t)$. In both cases, since $ub(t') - lb(t) > d(t)$, the new constraint intervals are non-empty. The complete description of the algorithm is given in the ISA Algorithm (see Algorithm 1).

Example 9 Consider again the set of tasks given in Fig. 2. The "unrefined" constraints on the tasks are computed as $C(t_1) = [0, 8]$, $C(t_3) = [2, 10]$, $C(t_5) = [0, 10]$ while $C(t_2) = [0, 0]$, $C(t_4) = [3, 3]$, $C(t_6) = [7, 7]$ and $C(t_7) = [9, 9]$. Since there is overlap between the constraints of task t_1, t_3 and t_3, t_5, these constraints have to be adapted. The result is the following set of constraints: $C(t_1) = [0, 4]$, $C(t_3) = [6, 7]$ and $C(t_5) = [0, 10]$. It is easily verifiable that all local schedules that adhere to their constraints are feasible and also that all such local schedules can be combined to obtain a global schedule that is correct and has a makespan of 11.

Algorithm 1 Generalised interval based scheduling (ISA)

Require: Partially ordered set of tasks $(T, \prec)$, for every task $t \in T$ its depth $depth(t)$ and its height $height(t)$;
Ensure: For every $t \in T$ its scheduling interval $C(t) = [lb(t), ub(t)]$;
1: $depth(T) := max_{t \in T}\{depth(t) + d(t)\}$
2: **for all** $t \in T$ **do**
3: $lb(t) := depth(t)$
4: $ub(t) := depth(T) - height(t)$
5: **for all** $t, t' \in T$ belonging to different agents such that $t \prec t'$ and $lb(t') - ub(t) < d(t)$ **do**
6: $ub(t) = lb(t) + \left\lfloor \frac{ub(t') - lb(t) - d(t)}{2} \right\rfloor$
7: $lb(t') = ub(t) + d(t)$
8: **for all** $t \in T$ **do**
9: return $C(t) = [lb(t), ub(t)]$

Note that in ISA, the removing overlapping intervals procedure corresponds to a simplified variant of Hunsberger's [26] decoupling algorithm for STNs.

3.1.2 The price of autonomy of autonomous scheduling

It is trivial to show that ISA runs in polynomial time. Each interval $[lb(t), ub(t)]$ computed by ISA is always non-empty and for every $t \in T$, $ub(t) \leq depth(T) - d(t)$. Moreover, for every pair t, t' of tasks, $t \prec t'$ implies $ub(t) + d(t) < lb(t')$. Hence, it is not difficult to see that (i) every local schedule σ_i satisfying the local constraints C_i will be a feasible schedule for the set of tasks T_i and (ii) the merge of every set $\{\sigma_i\}_{i=1}^n$ of local schedules is a feasible global schedule, thus ensuring the correctness and makespan optimality of the algorithm.

Proposition 1 *The interval-based scheduling algorithm (*ISA*) ensures a correct global schedule and it is efficient in terms of makespan.*

With respect to flexibility of this autonomous scheduling method, it is not difficult to see that it satisfies maximal flexibility. Here, we call a set $C = \{C(t) \mid t \in T\}$ of interval constraints for a scheduling instance $\langle \{T_i\}_{i=1}^n, \prec, c(), d() \rangle$ maximally flexible, if there does not exist any strict weakening[4] C' of C such that C' also allows for autonomous scheduling of $\langle \{T_i\}_{i=1}^n, \prec, c(), d() \rangle$ and is makespan efficient.

Proposition 2 *Any strict weakening the set C of constraints imposed by* ISA *either leads to a infeasible schedule or leads to a non optimal makespan.*

Proof See Appendix A. □

Summarising, we have the following property:

Theorem 1 ISA *ensures a maximally flexible set of constraints and a makespan efficient global schedule. It ensures the price of autonomy is 1, provided that the agents have unbounded concurrency.*

Note, however, that the minimal global makespan is ensured by the proposed algorithm ISA only under the assumption that the participating agents have capabilities to perform a potentially unbounded number of tasks at the same time. Often, this assumption is not realistic as agents may only have limited capacity to perform tasks. Therefore, in the next section, we study the case when every agent is capable of performing only a single task at any point in time (sequential agents).

[4] A set C' is a strict weakening of C if every schedule σ satisfying C also satisfies C' but not vice versa.

3.2 Coordinating agents with bounded concurrency

In this section, we adapt the method to accommodate for bounded concurrency requirements of agents. In particular, we consider the case where agents are strictly sequential. We then prove that in this latter case designing a makespan-efficient autonomous scheduling method is NP-hard. The good news, however, is that there exist good approximation algorithms for makespan efficient autonomous sequential scheduling, if we allow the tasks to be processed preemptively.

A scheduling instance $\langle \{T_i\}_{i=1}^n, \prec, c(), d() \rangle$ where $c(i) = 1$ for every A_i is called a sequential scheduling instance, abbreviated as $\langle \{T_i\}_{i=1}^n, \prec, 1, d() \rangle$. Like in the unbounded case, we would like to come up with a set C of constraints $C(t) = [lb(t), ub(t)]$ for each task $t \in T$ such that the agents are able to construct their *sequential* schedule independently from the others. Any individual schedule σ_i for a sequential agent A_i with the set of tasks T_i assigned to it, has to satisfy the following conditions:

- $lb(t) \leq \sigma_i(t) \leq ub(t)$ for every $t \in T_i$ where $C(t) = [lb(t), ub(t)]$;
- for every $t, t' \in T_i, t \neq t'$ implies $\sigma_i(t) - \sigma_i(t') \geq d(t)$ or $\sigma_i(t') - \sigma_i(t) \geq d(t')$.

The first condition ensures that schedules do not violate precedence constraints and the second ensures that at most a single task is scheduled in a given point of time. We design an additional set of constraints such that each agent can determine a sequential schedule for his set of tasks. While designing such constraints for autonomous scheduling if the agents are unbounded it is possible to develop constraints that ensure efficiency whereas, the equivalent problem for sequential agents turns out to be infeasible. The reason mainly is because we cannot ensure that, based on a given set of constraints delivered to the individual agents, they are able to find an efficient sequential schedule satisfying all the constraints. More precisely, while in the unbounded case we were able to find a minimum makespan M for the total set of tasks and could guarantee that given a set C of additional task constraints any set $\{\sigma_i\}_{i=1}^n$ of locally feasible schedules would result in a makespan M complying global schedule, finding such a makespan complying schedule in the sequential case is an intractable problem.

Proposition 3 *Given a sequential scheduling instance $\langle \{T_i\}_{i=1}^n, \prec, 1, d() \rangle$ and a positive integer M, the problem to decide whether there exists a set of constraints C such that the scheduling instance allows for a solution with make-span M by autonomous scheduling is NP-hard.*

Proof See Appendix B. □

Note that the complexity is quite independent upon the number of agents. Already two agents suffice to render the problem hard and, in particular, the problem derives its hardness from the difficulty to determine for a *single* agent the set of constraints that would allow it to determine its own schedule without violating the global makespan.

3.3 ISAS: ISA for sequential agents

Since a polynomial-time exact algorithm is not possible (unless $\mathsf{P} = \mathsf{NP}$) for autonomous scheduling in the sequential agent case, we have to rely on approximation algorithms. As we have shown in some recent work [59], there exists a polynomial 2-approximation algorithm for constructing a set of constraints in the sequential agent scheduling case if all the tasks $t \in T$ have unit durations $d(t) = 1$. This algorithm constructs a maximally flexible set of

constraints guaranteeing that the resulting global makespan is never more than twice the optimal makespan that can be realised by sequential scheduling agents. We will briefly discuss the outline of this algorithm and reuse it (after some adaptations) to the general sequential agent scheduling case.

The basic idea in this algorithm is to first use the ISA algorithm to determine the set of constraints C for the unit duration tasks. As we have shown above, if the agents would be able to handle tasks concurrently, an agent would be able to find a schedule satisfying all the constraints. In the sequential agent scheduling case, this might not be possible. For example, if there are three unrelated tasks t_1, t_2 and t_3 of unit duration, where the first two are given to agent A_1 and the third to agent A_2, agent A_1 is not able to schedule both tasks given the constraints $C(t_1) = C(t_2) = [0, 0]$. There is, however, a simple way to tell whether a given agent is able to find a sequential schedule for all tasks $t \in T_i$ with the constraints $C(t)$ given to it. Consider the bipartite graph $G_i = (T_i \cup N_i, E_i)$ where N_i is the set of all time points occurring in the intervals $C(t) = [lb(t), ub(t)]$ of tasks $t \in T_i$, and $(t, n) \in E_i$ iff $n \in C(t)$.[5] It is not difficult to see that there exists a sequential schedule for agent A_i iff the graph G_i has a *maximum matching* [13] that includes every task $t \in T_i$.

If the (polynomial) maximum matching algorithm is not able to find a complete matching for T_i, i.e., some of the tasks could not be scheduled, there must be a *scheduling conflict* between a task t in the matching and a task t' not in the matching. Such a conflict can be resolved by adding a new precedence constraint $t \prec t'$ between t and t' and calling the ISA algorithm again on the extended scheduling instance. Note that the result of such extensions of the precedence relation is twofold (i) the conflict between t and t' is removed and (ii) the global makespan $d(T)$ might be increased.

Continuing our last example mentioned above, consider the two tasks t_1 and t_2 agent A_1 was not able to handle. A maximum matching for G_1 contains either t_1 or t_2. If we add a precedence constraint $t_1 \prec t_2$ to the set of tasks, agent A_1 will receive the constraints $C(t_1) = [0, 0]$ and $C(t_2) = [1, 1]$ is able to find a suitable sequential schedule for its set of tasks.

This matching, extending the precedence relation, and calling the ISA algorithm is repeated until we are guaranteed that for each agent there exists at least one sequential schedule.[6] The result is a set of constraints C guaranteeing that any schedule resulting from independently chosen schedules realises a makespan that is at most twice as long as the optimal makespan.

One of the attractive features of the unit-duration sequential scheduling case is the existence of a polynomial decision procedure (the maximum matching algorithm) for deciding whether there exists a sequential schedule satisfying the constraints $C_i(t)$ for an agent A_i. The reduction from PARTITION given above shows that, unless P=NP, we cannot hope to find a solution for the same problem in the general sequential scheduling case.

There is, however, a possibility to reuse the approximation algorithm sketched above if we assume that, although the agents are strictly sequential, the tasks can be accomplished using *preemption*. This enables an agent to complete a part of task t, then to start some other tasks, process a next part of t and so on. If this is allowed, we can easily reduce a sequential scheduling instance $\langle \{T_i\}_{i=1}^n, \prec, 1, d() \rangle$ to a sequential scheduling instance with unit durations as follows.

Each task $t \in T$ is split in unit parts $t^1, \ldots, t^{d(t)}$, we add the constraints $t^j \prec t^{j+1}$ for $j = 1, \ldots, d(t) - 1$. Finally, every precedence constraint $t \prec t'$ is replaced by the constraint $t^{d(t)} \prec t'^1$. See Fig. 4 for an illustration. Note that this assumption implies that the sequential

[5] Note that we assume integer values for schedules $\sigma(t)$.

[6] This procedure must halt because conflicts can never reoccur.

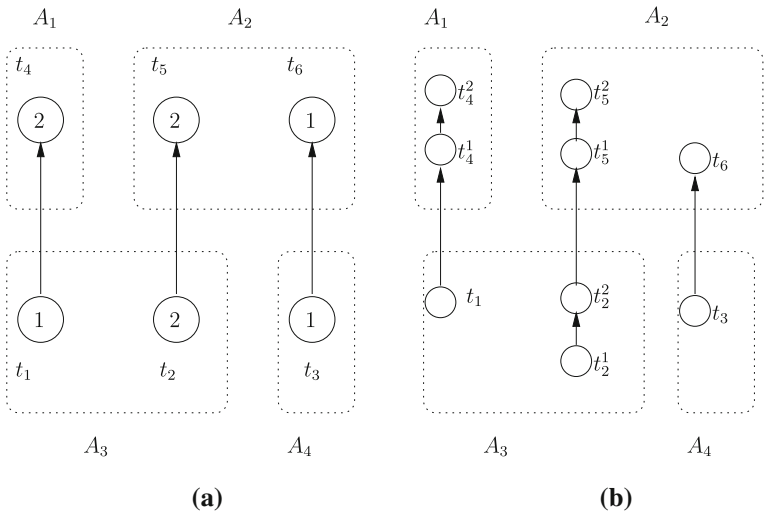

Fig. 4 A sequential scheduling instance (**a**) and its reduction to the equivalent unit-duration case (**b**) if preemption is allowed

scheduling case with arbitrary task durations can be reduced to the unity duration sequential case. Hence, we can reuse the approximation algorithm for this case, too, obtaining a 2-approximation algorithm for autonomous scheduling of tasks with arbitrary durations. There is of course a catch. The approximation algorithm is only polynomial for those instances where the durations $d(t)$ are not super-polynomial in the number of tasks $|T|$. Otherwise, the splitting of tasks in unit duration tasks would result in a super polynomial number of unit-duration tasks. The complete algorithm is described in Algorithm 2.

Algorithm 2 The matching based sequential adaptation of ISA for tasks with heterogenous durations (ISAS)

1: **Inputs:** constraint intervals $[lb(t), ub(t)]$ for all the tasks $t \in T$ as computed by ISA
2: **Outputs:** Revised $lb(t)$ and $ub(t)$ for each $t \in T$ to enable sequential scheduling
3: **for** each task t such that $d(t) > 1$ **do**
4: split t into a partial order $t^j \prec t^{j+1}$ where $j = 1, \ldots, d(t-1)$
5: replace every precedence constraint $t \prec t'$ by $t^{d(t)} \prec t'$
6: replace every precedence constraint $t' \prec t$ by $t' \prec t^1$
7: **while** there exists a (T_i, C_i) not allowing for a maximal matching M_i containing T_i **do**
8: take a task t contained in M_i and an overlapping task t' not occurring in M_i
9: add $t \prec t'$ or $t' \prec t$ to the partial order $(T, \prec)$
10: run ISA on the updated partial order $(T, \prec)$, update the set of constraint intervals C

Example 10 Consider the problem instance in Fig. 4. Here, the tasks are assigned to 4 agents $T_1 = \{t_4\}$, $T_2 = \{t_5, t_6\}$, $T_3 = \{t_1, t_2\}$ and $T_4 = \{t_3\}$with task durations $d(t_1) = d(t_3) = d(t_6) = 1$ and $d(t_2) = d(t_4) = d(t_5) = 2$. The precedence constraints are $t_1 \prec t_4$, $t_2 \prec t_5$ and $t_3 \prec t_6$. The minimal makespan for the unbounded case is $d(T) = 4$. This implies that after task splitting of task t_2, agent A_3 will receive the constraints $C(t_1) = [0, 1]$ and $C(t_2^1) = [0, 0]$ and $C(t_2^2) = [1, 1]$. If t_1 is included in the maximum matching (and t_2^1

is not), an additional constraint $t_1 \prec t_2^1$ is added to the set of tasks. As a result, the depth $depth(T) = 5$ and the new constraints are $C(t_1) = [0, 0], C(t_2^1) = [1, 1]$ and $C(t_2^2) = [2, 2]$. Continuing the procedure with the remaining agents in the same way can result in a joint schedule with makespan 6, where as the optimal makespan for sequential scheduling is 5.

The ISAS algorithm is a 2-approximation algorithm. Thus the worst-case makespan ratio of the joint schedule obtained by ISAS, versus the optimal schedule, is bounded by a factor of 2. Hence, we conclude that the price of autonomy of autonomous scheduling using ISAS algorithm is less than or equal to 2.

Proposition 4 *The price of autonomy of autonomous scheduling using the* ISAS *algorithm is at most 2.*

In this section we have seen so far that simple algorithms can be used to ensure concurrent decomposability and also guarantee efficient makespan. We have seen that the ISA which is based on the depth/height of the task in a task graph, captures the subset of all plans that ensure makespan efficiency as well as maximal flexibility. We have also seen that when agents are capable of performing a single task at a given point in time, makespan-efficiency suffers due to the inherent intractability of the problem. However, an important issue that features in many planning problems is the one involving shared resources. In this section we have assumed that all agents have "sufficient" resources to carry out their tasks. However, it is not necessarily true. In the next section we describe our research regarding the handling of resource constraints.

4 Sequential planning: coordination of resource usage

In this section, we concentrate on the resource coordination problem $(\mathcal{T}, R, C_R)$ in which the task constraints have been disregarded. Instead, we coordinate the usage of resources that the agents need to perform their tasks. The coordination should ensure that *the usage of a resource is always less than or equal to its capacity*. We can accomplish this by letting the agents make their (resource) plans in sequence, and by placing reservations on resources to reflect their planned usage.

In sequential planning, the first agent to make a plan is not constrained by any other agent, whereas the last agent to plan is faced with the reservations from all other agents. This raises the question of designing the order in which the agents should plan, for reasons of fairness (cf. [30]) and of global plan quality. With regard to global plan quality, the agent-ordering problem seems similar to the variable-ordering problem in constraint satisfaction problems (cf. [4]). A difference, however, is that in sequential planning we do not order single variables, but entire planning processes (which may update many variables).

Designing the agent ordering is not always relevant, or possible. In online settings, for example, agents arrive in a particular order, and so it is natural to apply a first come, first served approach to planning (i.e., the agent that arrives first also plans first). In such a setting, a relevant research issue is the *minimum perturbation problem* [2,41]: to find a good quality plan that makes few changes to existing plans. In this paper, however, we do not allow any changes to existing plans, and we investigate instead the effect of (arbitrary) agent ordering on global plan quality for a specific application domain.

The routing application we consider in this section is the transportation example of the introduction where task constraints are not taken into account. The resulting multi-agent routing problem has many applications, such as taxiway planning at airports [23,53], routing

of automated trucks at container terminals [17], and routing of Automated Guided Vehicles (AGV) in flexible manufacturing systems [9]. In this section, we will assume that each agent is assigned a single task, and no task is assigned to more than one agent.

We will start this section by presenting a model that describes how to translate agent plans to constraints for other agents. Next, we will present a single-agent routing algorithm that is optimal and runs in polynomial time. In Sect. 4.3, we will investigate three important questions about the sequential use of the optimal single-agent algorithm: (i) whether the price of autonomy is bounded above by a constant, (ii) whether the optimal agent ordering always leads to a multi-agent plan that is equal to the optimal plan, and (iii) what is the impact of the planning order on global plan quality?

4.1 A model for free time windows

In multi-agent route planning, the roads, locations, and intersections of the infrastructure are defined as a set of resources R. The resource constraint function C_R specifies resource constraints as follows. A resource r_i, $r_i \in R$, has a capacity $\mathrm{c}(r_i)$, denoting the maximum number of agents that can simultaneously make use of the resource, and a traversal time $\mathrm{d}(r_i) > 0$ which represents the minimum time it takes for an agent to traverse the resource. In addition to the capacity and the traversal constraints, we must specify that agents cannot use resources in any arbitrary order. Travel from resource r_i to resource r_j is possible if the pair (r_i, r_j) is in the *successor relation* $S \subseteq R \times R$.

An agent's plan consists of a sequence of resources, and a corresponding sequence of intervals in which to visit them.

Definition 4 (*Agent Plan*) An *agent plan* is a sequence $P = ((r_1, \tau_1), \ldots, (r_n, \tau_n))$ of n (resource, interval) pairs such that $\forall j \in [1, \ldots, n-1]$:

1. interval τ_j meets interval τ_{j+1},
2. $|\tau_j| \geq \mathrm{d}(r_j)$,
3. $(r_j, r_{j+1}) \in S$.

The first constraint in the above definition makes use of Allen's interval algebra [1],[7] and states that the exit time of the jth resource in the plan must be equal to the entry time into resource $j+1$. In other words, an agent must always occupy exactly one resource.

By making reservations on the relevant resources for every step in an agent's plan, we obtain a set of constraints for agents that still have to make a plan. From these constraints we infer the *free time windows* that are left on the resources.

Definition 5 (*Free Time Window*) Given resource r_i and a set of reservation intervals $\mathrm{Res}(r_i) = \{\tau_1, \ldots, \tau_m\}$ on resource r_i, a free time window on r_i is the largest interval $f = [\sigma, \phi]$ such that:

1. $\forall t \in f : |\{\tau_j \in \mathrm{Res}(r_i) | t \in \tau_j\}| < \mathrm{c}(r_i)$,
2. $(\phi - \sigma) \geq \mathrm{d}(r_i)$.

The above definition states that for an interval to be a free time window, there should not only be sufficient capacity at any moment during that interval (Condition 1), but it should also be long enough for an agent to traverse the resource (Condition 2). Note that the set of

[7] We make use of the *meets* predicate, which means that the end of one interval is equal to the start of the second, and the *precedes* predicate, which means that the end of one interval is earlier than the start of the second.

free time windows F_i on resource r_i is a vector $(f_{i,1}, \ldots, f_{i,\,m})$ of disjoint intervals such that for all $j \in [1, \ldots, m-1]$, $f_{i,j}$ precedes $f_{i,j+1}$.

Because it takes time to traverse a resource, one cannot enter a resource right at the end of a free time window. Therefore, we can associate an interval with the possible entry times and exit times of a free time window.

Definition 6 (*Free Time Window Entry and Exit Times*) Given a resource r_i and a free time window $f = [\sigma, \phi]$, the entry time window of f is given by $\tau_{\text{entry}}(f) = [\sigma, \phi - \text{d}(r_i)]$; the exit time window of f is given by $\tau_{\text{exit}}(f) = [\sigma + \text{d}(r_i), \phi]$.

To guarantee a feasible solution to the multi-agent routing problem, we need to decide what happens with an agent once it has reached its destination. If an agent stays at the destination location indefinitely, and two agents share the same unit-capacity destination resource, then no multi-agent plan can be found. Research on *idle vehicle positioning* [8] tries to figure out what to do with agents who are between jobs. However, a common approach is to make one of two simplifying assumptions: (i) all start and destination locations have infinite capacity (as in the work of [60]), or (ii) the agents enter the infrastructure at the start of their plan, leave it as soon as they have reached their destination (applied in the IPC04[8] airport planning domain, where aircraft land and take off [56]). We will make the latter assumption.

Under the assumption that agents are not in the infrastructure before or after executing their plans, it becomes easy to show that the sequential coordination method always yields a feasible multi-agent plan: if n agents have made their plans, then agent A_{n+1} can choose as its start time the latest exit time of all n agents. From that time point onwards, there is a free time window on every resource, so A_{n+1} will not come into conflict with any other agent.

4.2 Optimal single-agent routing

Our route planning algorithm is based on the idea of going from one free time window on one resource, to another free time window on a neighbouring resource. We now define when one free time window can be reached from another.

Definition 7 (*Free Time Window Reachability*) Given (i) a resource r_i, a free time window $f_{i,v} = [\sigma, \phi]$ on this resource, and a time $t \in \tau_{\text{exit}}(f_{i,v})$; (ii) a resource r_j and a free time window $f_{j,w}$ on resource r_j. We say that free time window $f_{j,w}$ is reachable from r_i at time t, denoted $f_{j,w} \in \rho(r_i, t)$, if:

1. $(r_i, r_j) \in S$,
2. $[t, \phi] \cap \tau_{\text{entry}}(f_{j,w}) \neq \varnothing$.

The reachability relation defines a graph of free time windows. Algorithm 3 finds the shortest path through the graph, which corresponds to the shortest-time, conflict-free route in the infrastructure.

Algorithm 3 works in the following way: in Line 2, we initialize the open list Q of free time windows to the start resource and the start time. For the simplicity of the specification of the algorithm, we specify partial plans only with (resource, free window entry time) pairs. In the actual implementation of the algorithm, an open list element has a backpointer to the open list element from which it was expanded, which is nil for the initial plan. In Line 4, we retrieve the open list element (r_i, t_i) with the lowest cost $t_i + \text{d}(r_i)$. Hence, the open list elements are sorted in order of increasing (minimum) *exit* times.

[8] International Planning Competition 2004.

Algorithm 3 Reservation-aware routing

Require: start resource r_s, destination resource r_d, start time t_s.
Ensure: shortest-time path from r_s to r_d.
1: **if** $\exists v \left[f_{s,v} \in F_s \mid t_s \in \tau_{\text{entry}}(f_{s,v}) \right]$ **then**
2: $Q \leftarrow \{(r_s, t_s)\}$
3: **while** $Q \neq \varnothing$ **do**
4: $(r_i, t_i) \leftarrow \text{argmin}_{(r,t)\in Q}(r + \text{d}(r))$
5: $Q \leftarrow Q \setminus \{(r_i, t_i)\}$
6: **if** $r_i = r_d$ **then**
7: **return** (r_i, t_i)
8: $t_{\text{exit}} \leftarrow t_i + \text{d}(r_i)$
9: **for all** $f_{j,v} \in \rho(r_i, t_{\text{exit}}) \cap \text{unmarked}(f_{j,v})$ **do**
10: $t_{\text{entry}} = \max(t_{\text{exit}}, \sigma_{j,v})$
11: $Q \leftarrow Q \cup (r_j, t_{\text{entry}})$
12: markWindow$(f_{j,v})$

To expand the current free time window, we consider, in Line 9, all (resource, free time window) pairs that are in $\rho(r_i, t_{\text{exit}})$, and which have not yet been expanded. The entry time into a reachable free time window $f_{j,v} = [\sigma_{j,v}, \phi_{j,v}]$ is either the entry time into the previous resource r_i plus the time it takes to traverse r_i, or, in case $f_{j,v}$ starts after $t_i + \text{d}(r_i)$, the start time $\sigma_{j,v}$ of $f_{j,v}$.

In Line 11 we simply add the new element to the open list Q, and, in Line 12, we mark the free time window $f_{j,v}$ as being visited, so it will not be expanded again. This is an important step, as it guarantees that we do not consider any free time window for expansion more than once.

Proposition 5 *Algorithm 3 returns an optimal solution.*

Proof See Appendix C. □

Algorithm 3 has a worst-case run-time that is polynomial in the number of free time windows. The number of free time windows, $|F|$, is at most $|\mathcal{A}| \cdot |R|$ under the assumption that we do not allow cyclic plans.

Proposition 6 *Algorithm 3 runs in* $O(|F| \log(|F|) + |F||R|)$ *time.*

Proof (Proof sketch) A free time window $f \in F$ can be put onto the open list Q at most once, and in every iteration of the while loop, one free time window is removed from Q (Line 4), Hence, the while loop is executed at most $|F|$ times, and none of the Lines 4–8 contribute more than $O(\log(|F|))$.

An important observation is that Lines 10–12 (inside the for-loop) are also executed at most $|F|$ times. In these lines, a free time window will be added to Q, and that can occur at most $|F|$ times.

Line 9 must determine for every $f_{j,v} \in \rho(f_{i,w}, x)$ (such that $x \in \tau_{\text{exit}}(f_{i,w})$) whether:

1. $f_{j,v}$ is reachable from t_{exit}, i.e., $f_{j,v} \in \rho(r_i, t_{\text{exit}})$, and
2. $f_{j,v}$ has not been visited yet, i.e., it is *unmarked*.

These checks require in an additional $O(|R|)$ computational overhead, which explains the factor $|F||R|$ in Algorithm 3's complexity. □

For a complete proof of the proposition, we refer the reader to [53].

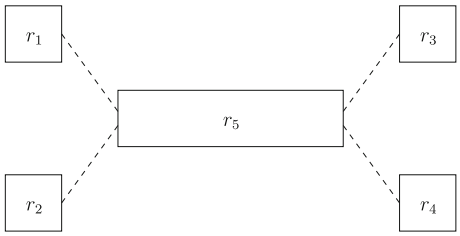

Fig. 5 An infrastructure with potential bottleneck resource r_5. *Rectangles* represent resources, *dashed lines* represent the connectivity relation S

To recap, we have presented a polynomial-time algorithm that an agent can use to find the shortest-time conflict-free route, given the constraints generated by previous agents. These constraints are encoded in the free time windows: every time an agent makes a reservation on a resource, it leaves a smaller free time window (or two free time windows) on that resource for subsequent agents. For the first agent to plan, all resources will have one free time window that starts at 0 and reaches to infinity. For agent A_n, there may be as many as $n + 1$ free time windows on a resource; more if we allow cyclic plans.

4.3 Price of autonomy in sequential routing

In sequential planning, many different planning orders are possible—for n agents, we have $n!$ possibilities—and which planning order is chosen can have a profound effect on the global plan (and its cost). If we assume that we make use of a free path routing algorithm that is optimal for a single agent, given the reservations of the other agents, then three important questions are:

1. Does there exist a fixed ratio k such that the global plan obtained by sequential, arbitrary order planning is at most k times the cost of the optimal global plan?
2. Is the best planning order guaranteed to result in the optimal global plan?
3. What is the impact of the planning order on global plan quality?

We will answer the first two questions in this section using examples. In Sect. 4.4, we will investigate the third question empirically. In answer to the first question, Example 11 shows (random order) sequential planning is not a fixed-ratio approximation scheme for the multi-agent routing problem: the worst-case global plan in Example 11 can be almost $\frac{|\mathcal{A}|}{2}$ times as bad as the optimal global plan.

Example 11 Consider the infrastructure of Fig. 5, and suppose that the set of agents is divided into two groups $\mathcal{A}_1$ and $\mathcal{A}_2$. All agents in $\mathcal{A}_1$ have their start location in r_1 and have r_3 as their destination location, whereas all agents in $\mathcal{A}_2$ start in r_4 and have r_2 as destination location. All resources are assumed to have infinite capacity and are bidirectional, but traffic is only allowed in one direction at the same time. We finally assume that $\mathrm{d}(r_5) > \mathrm{d}(r_1)+\mathrm{d}(r_2)+\mathrm{d}(r_3)+\mathrm{d}(r_4)$.

The optimal solution to the multi-vehicle routing problem is to let one group of agents plan before the other. In case the group $\mathcal{A}_1$ may plan first, then the last agent of $\mathcal{A}_1$ will arrive at resource r_3 at time

$$t_1 = \mathrm{d}(r_1) + \mathrm{d}(r_5).$$

At time t_1, agents from $\mathcal{A}_2$ will be able to enter r_5, and they will arrive at resource r_2 at time

$$t_2 = t_1 + \mathrm{d}(r_5).$$

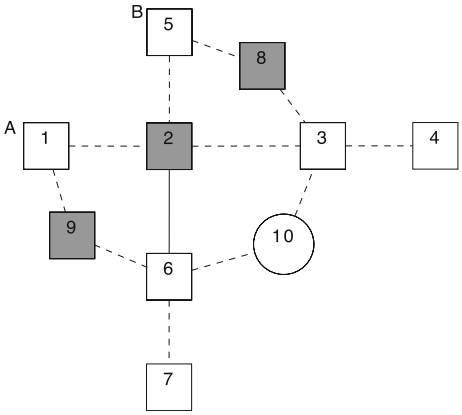

Fig. 6 *Shaded resources* have travel time 3, the *other resources* have travel time 1; the *circular resource* has capacity 2, the others have unit capacity. The minimum-makespan multi-agent plan is obtained when *both* agents A and B take a small detour

If planning alternates between groups, then every agent has to wait for the previous agent to clear resource r_5. Consequently, the final agent will arrive at its destination at time

$$t_3 > |\mathcal{A}| \cdot \mathrm{d}(r_5).$$

Since $\mathrm{d}(r_5)$ is the most significant contribution to the travel times, the alternating solution is almost $\frac{|\mathcal{A}|}{2}$ times as bad as the optimal solution.

Note that Example 11 does not provide a bound on the approximation ratio of sequential routing in general. However, it is very easy to show that we can always obtain a multi-agent plan that has a cost of $|A|$ times the optimal cost: we can simply let the start time of agent n be the end time of agent $n-1$. In this way, each agent can always choose the shortest path to its destination, because it will encounter no reservations of previous agents. Hence, the cost of the optimal plan will be at most $|A|$ times the cost of the $p^*_{\max}$, the cost of the most expensive single-agent plan. Since the optimal multi-agent plan must be at least as expensive as $p^*_{\max}$, the $|A|$-approximation result follows.

In the next example, we show that (optimal) sequential planning will not result in the optimal global plan, regardless of the planning order of the agents. To find the global plan with minimum make-span, both agents must choose a plan that is longer than their individually optimal plan.

Example 12 In Fig. 6, we have two agents A and B; agent A wants to go from r_1 to r_4, while agent B wants to go from r_5 to r_7. Note that in this figure, the shaded resources have travel time *three*; one resource, r_{10}, has capacity 2. The idea behind this example is that if either agent chooses the shortest route, then the other has to wait for it, whether it decides to take a detour or not. Again, assume that w.l.o.g. that agent A makes a plan first:

$$A : ((r_1, [0, 1]), (r_2, [1, 4]), (r_3, [4, 5]), (r_4, [5, 6])).$$

Agent B now has two reasonable options: to take the shortest path via r_2, or to make a detour along r_8 and r_3:

$$B_1 : \quad ((r_5, [0, 4]), (r_2, [4, 7]), (r_6, [7, 8]), (r_7, [8, 9]))$$
$$B_2 : ((r_5, [0, 1]), (r_8, [1, 5]), (r_3, [5, 6]), (r_{10}, [6, 7]), (r_6, [7, 8]), (r_7, [8, 9])).$$

Hence, agent B will have a finish time of 9. However, there exists a multi-agent plan with makespan 8:

$$A : ((r_1, [0, 1]), (r_9, [1, 4]), (r_6, [4, 5]), (r_{10}, [5, 6]), (r_3, [6, 7]), (r_4, [7, 8]))$$
$$B : ((r_5, [0, 1]), (r_8, [1, 4]), (r_3, [4, 5]), (r_{10}, [5, 6]), (r_6, [6, 7]), (r_7, [7, 8])).$$

For this multi-agent plan, both agents have a reservation on r_{10} in the interval [5, 6]. As r_2 is resource with capacity 2, this is allowed.

Note that although the above example is quite complicated, it only involves two agents. Simpler examples can be constructed if we allow resources to have only a single travel direction.

4.4 Sequential planning experiments

The goal of this section is to investigate the quality of a sequential route planning method, compared to a centralized solver that could, in theory, find an optimal multi-agent plan. Due to the intractability of finding an optimal (multi-agent) solution, we investigate instead whether the order in which agents plan has a big impact on plan quality.

In previous work [53], we tested our routing algorithm in a taxiway routing experiment on the infrastructure of Amsterdam Airport Schiphol. In those experiments we did not observe a big difference in global plan cost for different planning orders. In this section, we will extend our experiments to random graphs, and focus on the worst case difference between the best and the worst planning order.

4.4.1 Setup

For the experiments, we had 600 agents each with a single pair (r_s, r_d) of start location and destination location. For each type of infrastructure (more on the infrastructures shortly), we tried 100 different permutations of the agent planning order. We measured the difference between the order with the minimum makespan and the order with the maximum makespan. This difference is an indication of the 'price of sequential planning': how much more costly is the worst planning order compared to the optimal plan (or planning order)?

For infrastructures, we generated four sets of random graphs with different node degrees (we shall see in the figures that this is a relevant characteristic for the performance of a multi-agent routing algorithm). The edges of these graphs model 'lane' resources, and the nodes model intersection resources. The traversal time of a resource is a linear function of the length of the edge, in case of a lane resource, or 1 in case of an intersection resource. The average node degree ranges from around 4, to a little over 2. An average node degree of four corresponds to having about twice as many edges as there are nodes. Our choice for the maximum factor of 2 was motivated by the research into random *planar* networks by Denise et al. [16], who found that for approximately maximum random planar networks, the number of edges is around twice the number of nodes. Having planar networks makes sense in a transportation setting, but unfortunately constructing them is rather involved (cf. [6]), so we constructed ordinary random networks. An average node degree of around 2 corresponds to having roughly an equal number of nodes and edges, which constitutes a sparse but connected graph.

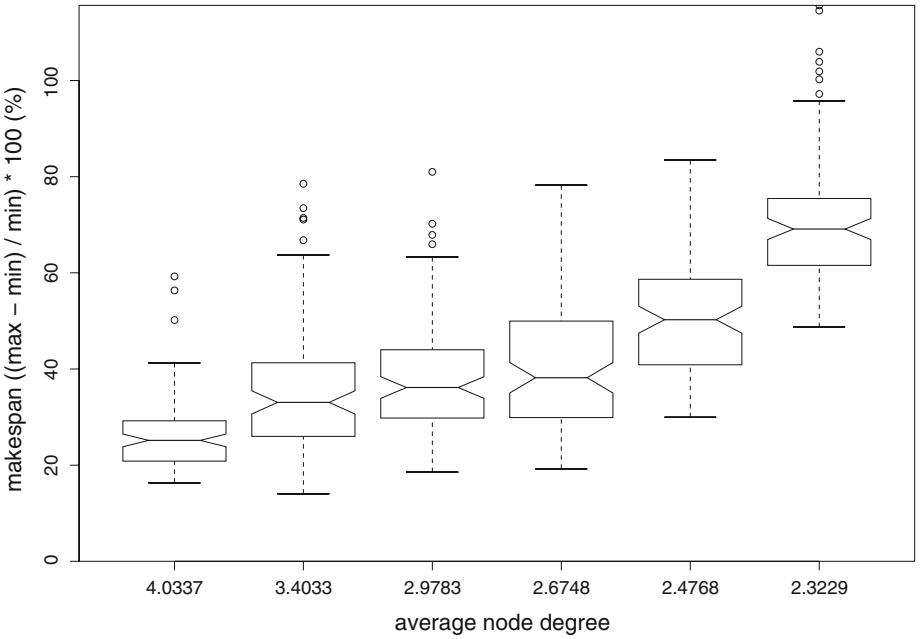

Fig. 7 Price of autonomy for different types of random graphs

4.4.2 Results

From Fig. 7 we can draw a number of interesting conclusions. The first is that makespan increases sharply as the average node degree of the infrastructure decreases. We believe there are at least two reasons for this occurrence. The first reason is exemplified by Example 11: if the average node degree is low, the infrastructure will consist of many 'corridors' of resources, i.e., sequences of resources from which an agent cannot change course. In fact, such a corridor could also be modelled as a single long resource, such as r_5 in Fig. 5. Randomly scheduling agents from either end of the corridor can lead to very bad results, as explained in the example.

A more general reason why infrastructures with low node degrees result in poor performance is that the single-agent routing algorithm needs to be able to select an alternative route if the shortest one is congested. For networks with a high node degree, more alternative routes are available, and who schedules which resource first becomes less crucial.

A second conclusion is that the difference between the best planning order and the worst planning order typically varies from around 30% of global plan quality (for networks with a node degree of around 4) to 70% (for networks with a node degree approaching 2). The worst-case behaviour varies from 60% to 120%, which corresponds to an approximation ratio of 2.2. Hence, the order in which the agent plan can have a large impact on the final plan quality.

5 Conclusion and future work

In this paper we have discussed the *coordination by design* approach for multi-agent planning problems. In this paradigm, dependencies between agents are resolved prior to planning, to

such an extent that the individually developed plans can always be merged into a feasible global plan. The advantage of this method is that no plan revision is required, and agents do not need to communicate with each other during planning.

We have presented two ways in which to achieve coordination by design: *concurrent decomposition*, in which all agents receive tasks with an additional set of constraints prior to planning and are allowed to plan concurrently, and *sequential decomposition*, in which the agents plan in a sequence, and the plan of one agent enforces additional constraints for all subsequent agents. We have studied these coordination methods in the context of the *price of autonomy*: the ratio between the worst-case (cost of a) global plan developed by coordinated, selfish agents, and the cost of an optimal global plan.

We have seen that concurrent decomposition can be beneficially applied when we only take into account constraints between tasks (such as precedence constraints), and do not concern ourselves with the shared resources the agents might need for task completion. For example, our ISA (Interval-based Scheduling Algorithm) will coordinate agents in such a way that the price of autonomy is 1 (that is, there is no loss of performance due to coordination and selfishness), in case agents have no bound on the number of tasks they can perform at the same time. If agents can only perform a single task at a time, then our ISAS (ISA for sequential agents) algorithm can still guarantee a price of autonomy of at most 2.

We have applied sequential decomposition to the case where resource constraints are taken into account, but constraints between tasks are ignored. For our application domain of multi-agent routing, we were able to show that random-order sequential agent planning cannot guarantee that the price of autonomy is bounded by a constant. Additionally, we have shown that there exist problems where the optimal solution cannot be found by sequential planning of selfish (in the sense of maximizing personal utility) agents. However, these results should not necessarily deter us, as some routing problems are simply too complex to allow fixed-ratio approximation algorithms. For example, if we extend our routing problem with the constraint that there should be an empty resource between any two agents, then this problem becomes PSPACE-hard [24,25]. Our experimental results show that the expected price of autonomy increases as the number of resources in the infrastructure decreases.

This paper focused on the investigation of two decomposition based coordination by design methods: concurrent coordination and sequential coordination. Unfortunately, these two methods are not capable of solving every task-based planning problem as defined in Sect. 2. Consider the transportation example presented in the introduction (Sect. 1.1). If both task and resource constraints in the example have to be respected, neither of the proposed coordination methods work: the concurrent decomposition approach fails to solve it because it cannot arbitrate the resource usage among agents, whereas the sequential decomposition method might get into a deadlock.

A naive approach to solve this problem would be to use a "mixed" coordination method. That is, we first apply a "concurrent" coordination algorithm that generates a total ordering of the tasks; then, we could sequentially plan for each of the tasks in turn. Although this method would guarantee a feasible global plan, it would not be a very satisfactory solution to our problem. First of all, it would take away most of the agents' autonomy that we have tried to preserve, and secondly, the price of autonomy (if we can still call it that) could be horrendously high. Therefore, it remains an open problem whether there exists a suitable decomposition-based coordination method for general multi-agent planning problems. We leave it as a challenging topic to address in the future.

In this paper, we determine the price of autonomy of using a coordination method by quantifying the *efficiency loss* due to the independent planning of the agents. Another interesting way to look at the quality of the coordination method is to establish the *price of coordination*

by considering the *autonomy loss* of the agents, quantifying the effects of imposing additional constraints enforced by the coordination method. When we take both measures into account in determining the quality of coordination methods, we will end up studying *Pareto-efficient* coordination mechanisms. In this way, for a specific coordination task, we can choose the best alternative, given a tolerable performance loss and a desired level of autonomy.

Another important issue with respect to autonomy is *fairness*. For example, in the concurrent coordination method, we try to globally add a minimum number of constraints to agents. However, such a set of constraints are not necessarily locally minimal to each agent. The fairness can be defined in many ways [7,38]. For instance, we can try to reduce the maximal number of constraints that agents receive. It will be very interesting to study whether it is possible to develop a fair coordination mechanism, which is able to impose the constraints on the agents in a fair way, yet still ensures a good system performance.

Acknowledgements We would like to thank the anonymous referees for their helpful comments and suggestions. They have been very useful in improving the paper. This research was supported in part by the Technology Foundation STW, applied science division of NWO, and the Ministry of Economic Affairs of the Netherlands, project number CSI4006. This work was part of the research project "Market based task coordination in multi-agent planning" which was financially supported by the Netherlands Organization for Scientific Research (NWO).

Appendix

A. Proof of Proposition 2

This proposition can easily be proven by noticing that the ISA algorithm generates a set of constraints $C = \{C(t) \mid t \in T\}$ such that (i) for every task t such that $succ(t) = \varnothing$ we have $ub(t) = depth(T) - d(t)$; (ii) for every t such that $pred(t) = \varnothing$ it holds that $lb(t) = 0$; (iii) for every pair of tasks t, t' belonging to different agents, such that $t \prec t'$ it holds that $lb(t') = ub(t) + d(t)$. (iv) For every pair of tasks t, t' belonging to the same agent, such that $t \prec t'$ we have $lb(t) + d(t) \leq lb(t')$ and $ub(t) + d(t) \leq ub(t')$.

This implies that any strict weakening C' of C would contain a constraint $C'(t) = [lb(t), ub(t)]$ such that either *(a)* $lb(t) < 0$ or *(b)* $ub(t) > depth(T) - d(t)$ or *(c)* there exists some t' such that $t \prec t'$ and $ub(t) + d(t) > lb(t')$ or *(d)* there exists some t' such that $t' \prec t$ and $ub(t') + d(t) > lb(t)$ or *(e)* there exists some t' such that $lb(t) + d(t) > lb(t')$ or *(f)* there exists some t' such that $ub(t) + d(t) > ub(t')$. Clearly, case *(a)* and *(b)* would imply that some C'-satisfying schedules are not make span efficient and if case c, d, e or f holds, some C'-satisfying schedules violate a precedence constraint. Hence, the proposition.

B. Proof of Proposition 3

We reduce the PARTITIONING problem [22] (Given a set S of integers, is there a subset S' of S such that $\sum_{s \in S'} s = \sum_{s \in \bar{S}} s$, where $\bar{S} = S - S'$?) to the autonomous scheduling for sequential agents problem.

Take an instance S of PARTITIONING and let $d_S = \sum_{s \in S} s$. Without loss of generality, we can assume d_S to be even. Consider the following set of tasks $T = \{t_s | s \in S\} \cup \{t_a, t_b, t_c\}$.

For every task $t_s \in T$, let $d(t_s) = s$, let $d(t_a) = \frac{d_S}{2}$, let $d(t_b) = 1$, $d(t_c) = \frac{d_S}{2} + 1$, and let $\prec = \{(t_a \prec t_b), (t_b \prec t_c)\}$. Furthermore, there are two agents A_1 and A_2, where A_1 has to perform the tasks t_a and t_c and agent A_2 has to perform all the remaining tasks $(T - \{t_a, t_c\})$. Finally, let $M = d_S + 1$.

If the agents are sequential, there exists a set of constraints C allowing for a make-span (M) efficient autonomous scheduling solution iff the PARTITION instance S has a solution: exactly in that case, agent A_2 is able to process one subset of its set of tasks in the interval $[0, d(t_a)]$, starts t_b in the interval $[d(t_a), d(t_a)]$ and completes the remaining subset of tasks in the interval $[d(t_a) + 1, d_S + 1]$.

C. Proof of Proposition 5

First of all, we prove by induction that for any $k \geq 0$ during the k-th execution of the while-loop algorithm's execution, each pair $(r_i, t_i) \in Q$ represents the earliest time to reach the free time window $f_{i,k}$ such that $t_i \in f_{i,k}$, having started from (r_s, t_s).

Initially, the open list contains only (r_s, t_s), and the induction hypothesis holds for $k = 0$. Suppose now that after $k \geq 0$ iterations of the while-loop, the pair (r_i, t_i) is retrieved from the open list in Line 4. Let $f_{j,y} \in \rho(r_i, t_{\text{exit}})$, and let $t_{\text{exit}} = t_i + \mathrm{d}(r_i)$. Now there are two cases to consider:

Case 1: $t_{\text{exit}} \leq \sigma_{j,y}$. In Line 10, the entry time into $f_{j,y}$ is determined to be $\sigma_{j,y}$. Clearly, the free time window $f_{j,y}$ can be entered no earlier than its start time $\sigma_{j,y}$, so the induction hypothesis also holds for the pair $(r_j, \sigma_{j,y})$ that is added to the queue.

Case 2: $t_{\text{exit}} > \sigma_{j,y}$. The entry time into $f_{j,y}$ will be t_{exit}. To see that no earlier entry time into $f_{j,y}$ is possible, note that $\forall k \neq i, (r_k, t_k) \in Q : t_{\text{exit}} \leq t_k + \mathrm{d}(r_k)$. Hence, for any pair (r_k, t_k) such that $f_{j,w} \in \rho(r_k, t_k)$, the entry time into $f_{j,y}$ would be larger than t_{exit}.

A second point to note is that there will be no iteration $m \geq k$ such that a pair (r_m, t_m) can be inserted into Q, such that $t_m + \mathrm{d}(r_m) < t_{\text{exit}}$. For all $(r_k, t_k) \in Q$, we have $t_{\text{exit}} \leq (t_k + \mathrm{d}(r_k))$, according to Line 4. If a new element (r_m, t_m) is inserted into the open list Q as a result of expanding (r_k, t_k), then $t_m \geq (t_k + \mathrm{d}(r_k))$, and, since travel times are greater than 0, $(t_m + \mathrm{d}(r_m)) > (t_k + \mathrm{d}(r_k)) \geq t_{\text{exit}}$.

Hence, there is no earlier entry time possible into window $f_{j,y}$ than t_{exit}, and the pair (r_j, t_{exit}) satisfies the induction hypothesis.

The proposition now follows since in each step of the algorithm, we expand a pair (r_i, t_i) to all free time windows reachable from the free time window determined by (r_i, t_i). Hence, we are guaranteed to find the first entry into the first reachable time window on destination resource r_d.

References

1. Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM, 26*(11), 832–843. doi:10.1145/182.358434.
2. Barták, R., Müller, T., & Rudová, H. (2003). A new approach to modelling and solving minimal perturbation problems. In *Recent advances in constraints* (pp. 223–249). New York, NY: Springer-Verlag.
3. Beaumont, O., Boudet, V., & Robert, Y. (2002). The iso-level scheduling heuristic for heterogeneous processors. In *Proceedings of 10th Euromicro workshop on parallel, distributed and network-based processing* (pp. 335–350). doi:10.1109/EMPDP.2002.994304.

4. Bessière, C., Chmeiss, A., & Sais, L. (2001). Neighborhood-based variable ordering heuristics for the constraint satisfaction problem. In *CP '01: Proceedings of the 7th international conference on principles and practice of constraint programming* (pp. 565–569). London, UK: Springer-Verlag.
5. Bird, R. (1984). The promotion and accumulation strategies in transformational programming. *ACM Transactions on Programming Languages and Systems, 6*(4), 487–504.
6. Bodirsky, M., Gröpl, C., & Kang, M. (2007). Generating labeled planar graphs uniformly at random. *Theoretical Computer Science, 379*(3), 377–386.
7. Brams, S. J., & Taylor, A. D. (1996). *Fair division: From cake cutting to dispute resolution*. New York, NY: Cambridge University Press.
8. Bruno, G., Ghiani, G., & Improta, G. (2000). Dynamic positioning of idle automated guided vehicles. *Journal of Intelligent Manufacturing, 11*(2), 209–215.
9. Buyurgan, N., Meyyappan, L., Saygin, C., & Dagli, C. H. (2007). Real-time routing selection for automated guided vehicles in a flexible manufacturing system. *Journal of Manufacturing Technology, 18*(2), 169–181.
10. Buzing, P. C., ter Mors, A. W., Valk, J. M., & Witteveen, C. (2006). Coordinating self-interested planning agents. *Autonomous Agents and Multi-Agent Systems, 12*(2), 199–218.
11. Clement, B., Barrett, A., Rabideau, G., & Durfee, E. (2001). Using abstraction in planning and scheduling. In *Proceedings of the sixth European conference on planning (ECP-01)* (pp. 145–156).
12. Clement, B. J., Durfee, E. H., & Barrett, A. C. (2007). Abstract reasoning for planning and coordination. *Journal of Artificial Intelligence Research, 28*, 453–515.
13. Cormen, T. T., Leiserson, C. E., & Rivest, R. L. (1990). *Introduction to algorithms*. Cambridge, MA, USA: MIT Press.
14. Cox, J. S., & Durfee, E. H. (2003). Discovering and exploiting synergy between hierarchical planning agents. In *Second international joint conference on autonomous agents and multiagent systems (AAMAS '03)* (pp. 281–288). New York, NY: ACM Press.
15. Decker, K. S., & Lesser, V. R. (1994). Designing a family of coordination algorithms. In *Proceedings of the thirteenth international workshop on distributed artificial intelligence (DAI-94)* (pp. 65–84). http://citeseer.nj.nec.com/decker95designing.html
16. Denise, A., Vasconcellos, M., & Welsh, D. (1996). The random planar graph. *Congressus Numerantium, 113*, 61–79.
17. Duinkerken, M. B., Ottjes, J. A., & Lodewijks, G. (2006). Comparison of routing strategies for AGV systems using simulation. In *WSC '06: Proceedings of the 38th conference on winter simulation* (pp. 1523–1530).
18. Durfee, E. H. (1999). Distributed problem solving and planning. In G. Weiß (Ed.), *Multiagent systems: A modern approach to distributed artificial intelligence* (pp. 121–164). Cambridge, MA, USA: MIT Press.
19. Durfee, E. H., & Lesser, V. R. (1991). Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics, 21*(5), 1167–1183. http://citeseer.nj.nec.com/durfee91partial.html
20. Ephrati, E., & Rosenschein, J. S. (1993). Multi-agent planning as the process of merging distributed sub-plans. In *Proceedings of the twelfth international workshop on distributed artificial intelligence (DAI-93)* (pp. 115–129). http://www.cs.huji.ac.il/labs/dai/papers.html
21. Foulser, D., Li, M., & Yang, Q. (1992). Theory and algorithms for plan merging. *Artificial Intelligence Journal, 57*(2–3), 143–182. http://www.cs.sfu.ca/isa/pubs/
22. Garey, M., & Johnson, D. (1979). *Computers and intractability—a guide to the theory of NP-completeness*. New York, NY: W.H. Freeman and Company.
23. Hatzack, W., & Nebel, B. (2001). The operational traffic problem: Computational complexity and solutions. In A. Cesta (Ed.), *Proceedings of the 6th European conference on planning (ECP'01)* (pp. 49–60). New York, NY: Springer-Verlag.
24. Hearn, R. A., & Demaine, E. D. (2005). Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science, 343*(1–2), 72–96.
25. Helmert, M. (2006). New complexity results for classical planning benchmarks. In *Proceedings of the sixteenth international conference on automated planning and scheduling (ICAPS 2006)* (pp. 52–61). AAAI Press.
26. Hunsberger, L. (2002a). Algorithms for a temporal decoupling problem in multi-agent planning. In *Proceedings of eighteenth national conference on artificial intelligence* (pp. 468–475). AAAI Press.
27. Hunsberger, L. (2002b). *Group decision making and temporal reasoning*. Ph.D. thesis, Harvard University, Cambridge, MA, USA.

28. Hunsberger, L. (2003). Distributing the control of a temporal network among multiple agents. In *Proceedings of the 2nd international joint conference on autonomous agents and multiagent systems (AAMAS)* (pp. 899–906). New York, NY, USA: ACM Press. doi:10.1145/860575.860621.
29. Jennings, N. R. (1993). Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review, 8*(3), 223–250.
30. Jonker, G., Dignum, F., & Meyer, J. J. (2007). Achieving cooperation among selfish agents in the air traffic management domain using signed money. In *Proceedings of the sixth international joint conference on autonomous agents and multiagent systems (AAMAS'07)* (pp. 1258–1260).
31. Korf, R. E. (1987). Planning as search: A quantitative approach. *Artifcial Intelligence, 33*(1), 65–68.
32. Lansky, A. (1990). Localized search for controlling automated reasoning. In *Proceedings of the DARPA workshop on innovative approaches to planning, scheduling and control* (pp. 115–125).
33. Lansky, A. L., & Getoor, L. C. (1995). Scope and abstraction: Two criteria for localized planning. In *Proceedings of the international joint conference on artificial intelligence* (pp. 1612–1618). Morgan Kaufmann.
34. Lesser, V., Decker, K., Wagner, T., Carver, N., Garvey, A., Horling, B., Neiman, D., Podorozhny, R., NagendraPrasad, M., Raja, A., Vincent, R., Xuan, P., & Zhang, X. (2004). Evolution of the GPGP/TA-EMS domain-independent coordination framework. *Autonomous Agents and Multi-Agent Systems, 9*(1), 87–143. http://mas.cs.umass.edu/paper/268
35. Li, J. (2007). *Strategic negotiation models for grid scheduling*. Ph.D. thesis, TU Dortmund.
36. Maheswaran, M., & Siegel, H. J. (1998). A dynamic matching and scheduling algorithm for heterogeneous computing systems. In *HCW '98: Proceedings of the seventh HCW* (57 pp.). IEEE Computer Society.
37. Moses, Y., & Tennenholtz, M. (1992). On computational aspects of artificial social systems. In *Proceedings of DAI-92* (pp. 267–284).
38. Moulin, H. (2004). *Fair division and collective welfare*. Cambridge, MA: MIT Press.
39. Nisan, N. (1999). Algorithms for selfish agents: Mechanism design for distributed computation. In *Proceedings of the 16th annual symposium on theoretical aspects of computer science* (pp. 1–15). New York, NY: Springer-Verlag.
40. Papadimitriou, C. (2001). Algorithms, games, and the internet. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on theory of computing* (pp. 749–753). New York, NY, USA: ACM Press. doi:10.1145/380752.380883.
41. Ran, Y., Roos, N., & van den Herik, J. (2002). Methods for repair based scheduling. In *Proceedings of the 21st workshop of the UK Planning and Scheduling Special Interest Group, PLANSIG*.
42. Roughgarden, T. (2005). *Selfish routing and the price of anarchy*. Cambridge, MA: MIT Press.
43. Roughgarden, T., & Tardos, E. (2002). How bad is selfish routing? *Journal of the ACM, 49*(2), 236–259. doi:10.1145/506147.506153.
44. Scerri, P., Pynadath, D., Johnson, L., Rosenbloom, P., Si, M., Schurr, N., & Tambe, M. (2003). A prototype infrastructure for distributed robot-agent-person teams. In *AAMAS '03: Proceedings of the second international joint conference on autonomous agents and multiagent systems* (pp. 433–440). New York, NY, USA: ACM Press. doi:10.1145/860575.860645.
45. Schurr, N., Marecki, J., Lewis, J. P., Tambe, M., & Scerri, P. (2005). The DEFACTO system: Training tool for incident commanders. In *AAAI* (pp. 1555–1562).
46. Sebastia, L., Onaindia, E., & Marzal, E. (2006). Decomposition of planning problems. *AI Communications, 19*(1), 49–81.
47. Shehory, O., & Kraus, S. (1998). Methods for task allocation via agent coalition formation. *Artificial Intelligence, 101*(1–2), 165–200.
48. Shoham, Y., & Tennenholtz, M. (1995). On social laws for artificial agent societies: Off-line design. *Artificial Intelligence, 73*(1–2), 231–252. http://citeseer.nj.nec.com/shoham95social.html
49. Sih, G. C., & Lee, E. A. (1990). Scheduling to account for interprocessor communication within interconnection-constrained processor networks. In *ICPP* (1) (pp. 9–16).
50. Sims, M., Corkill, D., & Lesser, V. (2004). Separating domain and coordination knowledge in multi-agent organizational design and instantiation. In *Proceedings of the AAAI-04 workshop on agent organizations: Theory and practice* (pp. 1–7). California: AAAI Press. http://mas.cs.umass.edu/paper/381
51. Steenhuisen, J., Witteveen, C., ter Mors, A. W., & Valk, J. M. (2006). Framework and complexity results for coordinating non-cooperative planning agents. In K. Fischer et al. (Eds.), *Proceedings of the 4th German conference on multi-agent system technologies (MATES)*. Lecture Notes in Artificial Intelligence (Vol. 4196, pp. 98–109). New York, NY: Springer-Verlag.
52. Stone, P., & Veloso, M. M. (2002). Layered learning and flexible teamwork in RoboCup simulation agents. In *RoboCup-99: Robot soccer world cup III*. Lecture Notes in Computer Science (Vol. 1856, pp. 495–508). New York, NY: Springer-Verlag.

53. ter Mors, A. W., Zutt, J., & Witteveen, C. (2007). Context-aware logistic routing and scheduling. In *Proceedings of the seventeenth international conference on automated planning and scheduling* (pp. 328–335). AAAI Press. http://dutiih.st.ewi.tudelft.nl/terMorsZuttWitteveen-icaps-2007.pdf
54. Tonino, J., Bos, A., de Weerdt, M., & Witteveen, C. (2002). Plan coordination by revision in collective agent based systems. *Artificial Intelligence, 142*, 121–145.
55. Topcuouglu, H., Hariri, S., & Wu, M. Y. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems, 13*(3), 260–274. doi:10.1109/71.993206.
56. Trüg, S., Hoffmann, J., & Nebel, B. (2004). Applying automatic planning systems to airport ground-traffic control—a feasibility study. In *KI* (pp. 183–197).
57. von Martial, F. (1992). *Coordinating plans of autonomous agents*. Lecture Notes on Artificial Intelligence (vol. 610). Berlin: Springer-Verlag.
58. Walsh, W., Wellman, M., Wurman, P., & MacKie-Mason, J. (1998). Some economics of market-based distributed scheduling. In *Proceedings of 18th international conference on distributed computing systems* (pp. 612–621). doi:10.1109/ICDCS.1998.679848.
59. Yadati, C., Witteveen, C., Zhang, Y., Wu, M., & Putre, H. L. (2008). Autonomous scheduling. In *Proceedings of the FCS 2008* (pp. 73–79).
60. Zutt, J., & Witteveen, C. (2006). Operational transport planning with incidents, experiments with traplas. In H. J. van Zuylen (Ed.), *Proceedings of the 9th TRAIL congress (TRAIL'06)*.